ADST/WDL/TR-93-W003092; 31 March 1993
VOLUME 1 of 3; Rev. 0.0

SOFTWARE MAINTENANCE MANUAL

FOR THE

AIRNET AEROMODEL AND

WEAPONS MODEL CONVERSION

VOLUME 1 of 3

Rev. 0.0: 31 March 1993

CONTRACT NO. N61339-91-D-0001

D.O.: 0014

CDRL SEQUENCE NO. A005

Prepared for:

STRICOM
Simulator Training and
Instrumentation Command

Simulator Training and Instrumentation Command
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3275

Prepared by:

LORAL

ADST Program Office
12443 Research Parkway, Suite 303
Orlando, FL 32826

REPORT DOCUMENTATION PAGE

Form approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
31 March 1993

3. REPORT TYPE AND DATES COVERED
Version 0.0

4. TITLE AND SUBTITLE
Advanced Distributed Simulation Technology Software Maintenance Manual for the AIRNET Aeromodel and Weapons Model Conversion; Volume 1 of 3

5. FUNDING NUMBERS
Contract No. N61339-91-D-0001

6. AUTHOR(S)
Branson, Roger; McCarter, Steve

Accesion For		
NTIS CRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Loral Systems Company
ADST Program Office
12443 Research Parkway, Suite 303
Orlando, FL 32826

8. PERFORMING ORGANIZATION REPORT NUMBER
ADST/WDL/TR-93-W003092
CDRL A005

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Simulation, Training and Instrumentation Command
STRICOM
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3275

10. SPONSORING ORGANIZATION REPORT

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (Maximum 200 words)

The ADST Software Maintenance Manual provides guidance for modifying the aeromodel and weapons model data files.

14. SUBJECT TERMS

15. NUMBER OF PAGES
689

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	17. SECURITY CLASSIFICATION OF THIS PAGE	17. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

LIST OF TABLES

1. Scope.

This section shall be divided into the following paragraphs.

1.1. Identification.

This Software Maintenance Manual (SMM) applies to document number WDL/TR92-003011 entitled System Specification for the Rotary Wing Aircraft AirNet Aeromodel and Weapons Model Conversion. This SMM also applies to the AirNet CSCI.

1.2. System overview.

The Rotary Wing Aircraft (RWA) system and SIMNET Computer System Configuration Item (CSCI) simulates a manned flight vehicle and associated weapons systems for conducting simulated missions within a controlled database and tactical environment.

1.3. Document overview.

The following paragraphs and subparagraphs identify the data variables, Computer Software Units (CSU) and algorithms that use the data read from data files during initialization. These data files were constructed as a task of the Rotary Wing Aircraft AirNET Aeromodel and Weapons Model Conversion Delivery Order and are documented in detail in Software Design Document for the AirNET Aeromodel and Weapons Model Conversion. Certain CSUs were modified to allow the reading of data values from data files. Computer Software Components (CSC) and CSUs existing in original code are only documented herein to the extent that the data from these data files is used. The original function and operation of the software was not modified. This additional capability allows for the change of variables without requiring a recompile. This SMM is compiled as a guide to the software programmer to assist in understanding how the data variables are used and how a modification of the data will effect computation of certain performance characteristics and limits of the aeromodel, engines, and weapon systems. The modifications to the MCC and communications software are covered in a separate volume.

2. Referenced documents.

The following documents are referenced within this document.

WDL/TR--92-003011	SYSTEM SPECIFICATION FOR THE ROTARY WING AIRCRAFT AIRNET AEROMODEL AND WEAPONS MODEL CONVERSION, 6 JUNE 1992.
WDL/TR--93-003036	SOFTWARE DESIGN DOCUMENT FOR THE ROTARY WING AIRCRAFT AIRNET AEROMODEL AND WEAPONS MODEL CONVERSION, 22 JANUARY 1993.

3. Modifiable data.

The following subparagraphs address the data contained in the data files which are read during initialization. Changes to the data files do not normally require a recompilation of the source code and re-link of the libraries. The configuration management group should be contacted if it is necessary to make source code changes to the baseline.

3.1 Aero_data

This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control.

3.1.1 MOMENT_OF_INERTIA_X

MOMENT_OF_INERTIA_X is a constant defining the moment of inertia of the vehicle in the x-axis.

3.1.1.1 Initialization

The constant MOMENT_OF_INERTIA_X is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MOMENT_OF_INERTIA_X          aero_data[ 0]
```

3.1.1.2 Usage

During real-time execution, this variable is not recomputed.

3.1.1.2.1 Algorithm

The MOMENT_OF_INERTIA_X is used to initialize the [1][1] element of the inertia matrix in the CSU aerodyn_init.

```
inertia_matrix[1] [1] = MOMENT_OF_INERTIA_X;
```

The MOMENT_OF_INERTIA_X is used to compute forces in the CSC aerodyn_simple_simul.

```
/* First, compute the angular velocity necessary to achieve the */
/* desired orientation in exactly one tick. (delta theta/ delta T) */
/* Then get the angular acceleration needed to get to that velocity */
/* In one tick.*/
for (i = X; i <= Z; ++i)
{
    vec_ptr[i] = ((des_ptr[i] - cur_ptr[i]) / DELTA_T / H_K1);
    angular_accel[i] = (vec_ptr[i] - angular_velocity_vector[i])
                / DELTA_T;
    res_ptr[i] = MOMENT_OF_INERTIA_X * angular_accel[i];
}
```

See APPENDIX B for a complete source code listing.

3.1.2 MOMENT_OF_INERTIA_Y

MOMENT_OF_INERTIA_Y is a constant defining the moment of inertia of the vehicle in the y-axis.

3.1.2.1 Initialization

The constant MOMENT_OF_INERTIA_Y is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MOMENT_OF_INERTIA_Y             aero_data[ 1]
```

3.1.2.2 Usage

During real-time execution, this variable is not recomputed.

3.1.2.2.1 Algorithm

The MOMENT_OF_INERTIA_Y is used to initialize the [2][2] element of the inertia matrix in the CSU aerodyn_init.

```
inertia_matrix[2] [2] = MOMENT_OF_INERTIA_Y;
```

See APPENDIX B for a complete source code listing.

3.1.3 MOMENT_OF_INERTIA_Z

MOMENT_OF_INERTIA_Z is a constant defining the moment of inertia of the vehicle in the z-axis.

3.1.3.1 Initialization

The constant MOMENT_OF_INERTIA_Z is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MOMENT_OF_INERTIA_Z                aero_data[ 2]
```

3.1.3.2 Usage

During real-time execution, this variable is not recomputed.

3.1.3.2.1 Algorithm

The MOMENT_OF_INERTIA_Z is used to initialize the [3][3] element of the inertia matrix in the CSU aerodyn_init.

```
inertia_matrix[3] [3] = MOMENT_OF_INERTIA_Z;
```

See APPENDIX B for a complete source code listing.

3.1.4 AIRFRAME_MASS

AIRFRAME_MASS is a constant defining the empty weight of the vehicle, especially, not including expendable items.

3.1.4.1 Initialization

The constant AIRFRAME_MASS is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define AIRFRAME_MASS                              aero_data[ 3]
```

3.1.4.2 Usage

During real-time execution, this variable is not recomputed.

3.1.4.2.1 Algorithm

The AIRFRAME_MASS is used to initialize the vehicle mass in the CSU aerodyn_init.

```
vehicle_mass_init (AIRFRAME_MASS + ORDINANCE_MASS,
                   inertia_matrix );
```

The AIRFRAME_MASS is used to update the vehicle gross weight by a call to the CSU compute_gross_weight.

```
vehicle_mass = AIRFRAME_MASS + ORDINANCE_MASS +
   fuel_get_current_level() * KILOGRAMS_PER_GALLON;/* kg */

gross_weight = vehicle_mass * GRAV_CONSTANT;    /* N */
```

See APPENDIX B for a complete source code listing.

3.1.5 ORDINANCE_MASS

ORDINANCE_MASS is a constant defining the weight of the ordinance on board the vehicle, especially, the expendables.

3.1.5.1 Initialization

The constant ORDINANCE_MASS is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  ORDINANCE_MASS                    aero_data[ 4]
```

3.1.5.2 Usage

During real-time execution, this variable is not recomputed.

3.1.5.2.1 Algorithm

The ORDINANCE_MASS is used to initialize the vehicle mass in the CSU aerodyn_init.

```
vehicle_mass_init (AIRFRAME_MASS + ORDINANCE_MASS,
                   inertia_matrix );
```

The ORDINANCE_MASS is used to update the vehicle gross weight by a call to the CSU compute_gross_weight.

```
vehicle_mass = AIRFRAME_MASS + ORDINANCE_MASS +
   fuel_get_current_level() * KILOGRAMS_PER_GALLON;/* kg */

gross_weight = vehicle_mass * GRAV_CONSTANT;   /* N */
```

See APPENDIX B for a complete source code listing.

3.1.6 GRAV_CONSTANT

GRAV_CONSTANT is a constant defining the gravitational constant.

3.1.6.1 Initialization

The constant GRAV_CONSTANT is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  GRAV_CONSTANT                     aero_data[ 5]
```

3.1.6.2 Usage

During real-time execution, this variable is not recomputed.

3.1.6.2.1 Algorithm

The GRAV_CONSTANT is used to update the vehicle gross weight by a call to the CSU compute_gross_weight.

```
gross_weight = vehicle_mass * GRAV_CONSTANT;    /* N */
```

See APPENDIX B for a complete source code listing.

3.1.7 CG_AC_X

CG_AC_X is a constant defining the location of the aircraft center of gravity in the x-axis.

3.1.7.1 Initialization

The constant CG_AC_X is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define CG_AC_X                                   aero_data[ 6]
```

3.1.7.2 Usage

During real-time execution, this variable is not recomputed.

3.1.7.2.1 Algorithm

The CG_AC_X is used to initialize the vehicle location of the aircraft center of gravity in the x-axis in the CSU aerodyn_init.

```
loc_ac_cg[X] = CG_AC_X;
```

See APPENDIX B for a complete source code listing.

3.1.8 CG_AC_Y

CG_AC_Y is a constant defining the location of the aircraft center of gravity in the y-axis.

3.1.8.1 Initialization

The constant CG_AC_Y is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define CG_AC_Y                                   aero_data[ 7]
```

3.1.8.2 Usage

During real-time execution, this variable is not recomputed.

3.1.8.2.1 Algorithm

The CG_AC_Y is used to initialize the vehicle location of the aircraft center of gravity in the y-axis in the CSU aerodyn_init.

```
loc_ac_cg[Y] = CG_AC_Y;
```

See APPENDIX B for a complete source code listing.

3.1.9 CG_AC_Z

CG_AC_Z is a constant defining the location of the aircraft center of gravity in the z-axis.

3.1.9.1 Initialization

The constant CG_AC_Z is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define CG_AC_Z                                   aero_data[ 8]
```

3.1.9.2 Usage

During real-time execution, this variable is not recomputed.

3.1.9.2.1 Algorithm

The CG_AC_Z is used to initialize the vehicle location of the aircraft center of gravity in the z-axis in the CSU aerodyn_init.

```
loc_ac_cg[Z] = CG_AC_Z;
```

See APPENDIX B for a complete source code listing.

3.1.10 VIRTUAL_WING_AREA

VIRTUAL_WING_AREA is a constant defining the effective wing area of the vehicle.

3.1.10.1 Initialization

The constant VIRTUAL_WING_AREA is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define VIRTUAL_WING_AREA                         aero_data[ 9]
```

3.1.10.2 Usage

During real-time execution, this variable is not recomputed.

3.1.10.2.1 Algorithm

The VIRTUAL_WING_AREA is used to compute the total lift of the virtual wing of the vehicle by a call to the CSU compute_lift_drag_forces.

```
lift_virtual_wing = dynamic_pressure *
        lift_coefficient_virtual_wing * VIRTUAL_WING_AREA;
```

See APPENDIX B for a complete source code listing.

3.1.11 VIRTUAL_WING_COP_AC_X

VIRTUAL_WING_COP_AC_X is a constant defining the location in the x-axis of the virtual wing center of pressure.

3.1.11.1 Initialization

The constant VIRTUAL_WING_COP_AC_X is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  VIRTUAL_WING_COP_AC_X          aero_data[10]
```

3.1.11.2 Usage

During real-time execution, this variable is not recomputed.

3.1.11.2.1 Algorithm

The VIRTUAL_WING_COP_AC_X is initialize the location in the x-axis of the center of pressure for the virtual wing in the CSU aerodyn_init.

```
loc_ac_virtual_wing_cop[X] = VIRTUAL_WING_COP_AC_X;
```

The location of the center of pressure for the virtual wing is used to compute lift and moments due to lift.

```
vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                moment_body_virtual_wing);
```

See APPENDIX B for a complete source code listing.

3.1.12 VIRTUAL_WING_COP_AC_Y

VIRTUAL_WING_COP_AC_Y is a constant defining the location in the y-axis of the virtual wing center of pressure.

3.1.12.1 Initialization

The constant VIRTUAL_WING_COP_AC_Y is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  VIRTUAL_WING_COP_AC_Y                    aero_data[11]
```

3.1.12.2 Usage

During real-time execution, this variable is not recomputed.

3.1.12.2.1 Algorithm

The VIRTUAL_WING_COP_AC_Y is initialize the location in the y-axis of the center of pressure for the virtual wing in the CSU aerodyn_init.

```
loc_ac_virtual_wing_cop[Y] = VIRTUAL_WING_COP_AC_Y;
```

The location of the center of pressure for the virtual wing is used to compute lift and moments due to lift.

```
vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                     moment_body_virtual_wing);
```

See APPENDIX B for a complete source code listing.

3.1.13 VIRTUAL_WING_COP_AC_Z

VIRTUAL_WING_COP_AC_Z is a constant defining the location in the z-axis of the virtual wing center of pressure.

3.1.13.1 Initialization

The constant VIRTUAL_WING_COP_AC_Z is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define VIRTUAL_WING_COP_AC_Z                    aero_data[12]
```

3.1.13.2 Usage

During real-time execution, this variable is not recomputed.

3.1.13.2.1 Algorithm

The VIRTUAL_WING_COP_AC_Z is initialize the location in the z-axis of the center of pressure for the virtual wing in the CSU aerodyn_init.

```
loc_ac_virtual_wing_cop[Z] = VIRTUAL_WING_COP_AC_Z;
```

The location of the center of pressure for the virtual wing is used to compute lift and moments due to lift.

```
vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
               moment_body_virtual_wing);
```

See APPENDIX B for a complete source code listing.

3.1.14 WING_LIFT_COEFFICIENT_FIT_3

WING_LIFT_COEFFICIENT_FIT_3 is a constant defining the fourth coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

3.1.14.1 Initialization

The constant WING_LIFT_COEFFICIENT_FIT_3 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and

is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  WING_LIFT_COEFFICIENT_FIT_3        aero_data[13]
```

3.1.14.2 Usage

During real-time execution, this variable is not recomputed.

3.1.14.2.1 Algorithm

The WING_LIFT_COEFFICIENT_FIT_3 is used to compute the unit lift of the virtual wing by a call to the CSU virtual_wing_lift_coefficient. The call to this CSU is commented out.

```
if (alpha > WING_STALL_AOA || alpha < 0.0)
    return (0.0);
else
 return   (((WING_LIFT_COEFFICIENT_FIT_3 *  alpha +
            WING_LIFT_COEFFICIENT_FIT_2) * alpha +
            WING_LIFT_COEFFICIENT_FIT_1) * alpha +
            WING_LIFT_COEFFICIENT_FIT_0);
```

See APPENDIX B for a complete source code listing.

3.1.15 WING_LIFT_COEFFICIENT_FIT_2

WING_LIFT_COEFFICIENT_FIT_2 is a constant defining the third coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

3.1.15.1 Initialization

The constant WING_LIFT_COEFFICIENT_FIT_2 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define WING_LIFT_COEFFICIENT_FIT_2        aero_data[14]
```

3.1.15.2 Usage

During real-time execution, this variable is not recomputed.

3.1.15.2.1 Algorithm

The WING_LIFT_COEFFICIENT_FIT_2 is used to compute the unit lift of the virtual wing by a call to the CSU virtual_wing_lift_coefficient. The call to this CSU is commented out.

```
if (alpha > WING_STALL_AOA || alpha < 0.0)
    return (0.0);
else
 return   (((WING_LIFT_COEFFICIENT_FIT_3 *  alpha +
            WING_LIFT_COEFFICIENT_FIT_2) * alpha +
            WING_LIFT_COEFFICIENT_FIT_1) * alpha +
            WING_LIFT_COEFFICIENT_FIT_0);
```

See APPENDIX B for a complete source code listing.

3.1.16 WING_LIFT_COEFFICIENT_FIT_1

WING_LIFT_COEFFICIENT_FIT_1 is a constant defining the second coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

3.1.16.1 Initialization

The constant WING_LIFT_COEFFICIENT_FIT_1 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define WING_LIFT_COEFFICIENT_FIT_1        aero_data[15]
```

3.1.16.2 Usage

During real-time execution, this variable is not recomputed.

3.1.16.2.1 Algorithm

The WING_LIFT_COEFFICIENT_FIT_1 is used to compute the unit lift of the virtual wing by a call to the CSU virtual_wing_lift_coefficient. The call to this CSU is commented out.

```
if (alpha > WING_STALL_AOA || alpha < 0.0)
    return (0.0);
else
 return   (((WING_LIFT_COEFFICIENT_FIT_3 *  alpha +
           WING_LIFT_COEFFICIENT_FIT_2) * alpha +
           WING_LIFT_COEFFICIENT_FIT_1) * alpha +
           WING_LIFT_COEFFICIENT_FIT_0);
```

See APPENDIX B for a complete source code listing.

3.1.17 WING_LIFT_COEFFICIENT_FIT_0

WING_LIFT_COEFFICIENT_FIT_0 is a constant defining the first coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

3.1.17.1 Initialization

The constant WING_LIFT_COEFFICIENT_FIT_0 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  WING_LIFT_COEFFICIENT_FIT_0        aero_data[16]
```

3.1.17.2 Usage

During real-time execution, this variable is not recomputed.

3.1.17.2.1 Algorithm

The WING_LIFT_COEFFICIENT_FIT_0 is used to compute the unit lift of the virtual wing by a call to the CSU virtual_wing_lift_coefficient. The call to this CSU is commented out.

```
if (alpha > WING_STALL_AOA || alpha < 0.0)
    return (0.0);
else
 return   (((WING_LIFT_COEFFICIENT_FIT_3 *  alpha +
           WING_LIFT_COEFFICIENT_FIT_2) * alpha +
           WING_LIFT_COEFFICIENT_FIT_1) * alpha +
           WING_LIFT_COEFFICIENT_FIT_0);
```

See APPENDIX B for a complete source code listing.

3.1.18 WING_STALL_AOA

WING_STALL_AOA is a constant defining the wing stall angle of attack.

3.1.18.1 Initialization

The constant WING_STALL_AOA is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  WING_STALL_AOA            (deg_to_rad(aero_data[17]))
```

3.1.18.2 Usage

During real-time execution, this variable is not recomputed.

3.1.18.2.1 Algorithm

The WING_STALL_AOA is used to control the computation of the unit lift coefficient of the virtual wing by a call to the CSU virtual_wing_lift_coefficient. The call to this CSU is commented out.

```
if (alpha > WING_STALL_AOA || alpha < 0.0)
    return (0.0);
else
 return   (((WING_LIFT_COEFFICIENT_FIT_3 *  alpha +
           WING_LIFT_COEFFICIENT_FIT_2) * alpha +
           WING_LIFT_COEFFICIENT_FIT_1) * alpha +
           WING_LIFT_COEFFICIENT_FIT_0);
```

See APPENDIX B for a complete source code listing.

3.1.19 VSTAB_AREA

VSTAB_AREA is a constant defining the effective vertical stabilator area.

3.1.19.1 Initialization

The constant VSTAB_AREA is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  VSTAB_AREA                    aero_data[18]
```

3.1.19.2 Usage

During real-time execution, this variable is not recomputed.

3.1.19.2.1 Algorithm

VSTAB_AREA is used to compute the total lift of the vertical stabilator by a call to CSU compute_lift_drag_forces.

```
lift_vstab = dynamic_pressure * lift_coefficient_vstab * VSTAB_AREA;
```

See APPENDIX B for a complete source code listing.

3.1.20 VSTAB_COP_AC_X

VSTAB_COP_AC_X is a constant defining the location in the x-axis of the center of pressure of the vertical stabilator for the vehicle.

3.1.20.1 Initialization

The constant VSTAB_COP_AC_X is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define VSTAB_COP_AC_X                          aero_data[19]
```

3.1.20.2 Usage

During real-time execution, this variable is not recomputed.

3.1.20.2.1 Algorithm

VSTAB_COP_AC_X is used to initialize the location in the x-axis of the center of pressure for the vertical stabilator in the CSU aerodyn_init.

```
loc_ac_vstab_cop[X] = VSTAB_COP_AC_X;
```

The loc_ac_vstab_cop vector is then used to compute the body forces and moments by a call to the CSC sum_body_forces_and_moments_about_ac.

```
vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
```

See APPENDIX B for a complete source code listing.

3.1.21 VSTAB_COP_AC_Y

VSTAB_COP_AC_Y is a constant defining the location in the y-axis of the center of pressure of the vertical stabilator for the vehicle.

3.1.21.1 Initialization

The constant VSTAB_COP_AC_Y is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define VSTAB_COP_AC_Y                          aero_data[20]
```

3.1.21.2 Usage

During real-time execution, this variable is not recomputed.

3.1.21.2.1 Algorithm

VSTAB_COP_AC_Y is used to initialize the location in the y-axis of the center of pressure for the vertical stabilator in the CSU aerodyn_init.

```
loc_ac_vstab_cop[Y] = VSTAB_COP_AC_Y;
```

The loc_ac_vstab_cop vector is then used to compute the body forces and moments by a call to the CSC sum_body_forces_and_moments_about_ac.

```
vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
```

See APPENDIX B for a complete source code listing.

3.1.22 VSTAB_COP_AC_Z

VSTAB_COP_AC_Z is a constant defining the location in the z-axis of the center of pressure of the vertical stabilator for the vehicle.

3.1.22.1 Initialization

The constant VSTAB_COP_AC_Z is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  VSTAB_COP_AC_Z                    aero_data[21]
```

3.1.22.2 Usage

During real-time execution, this variable is not recomputed.

3.1.22.2.1 Algorithm

VSTAB_COP_AC_Z is used to initialize the location in the z-axis of the center of pressure for the vertical stabilator in the CSU aerodyn_init.

```
loc_ac_vstab_cop[Z] = VSTAB_COP_AC_Z;
```

The loc_ac_vstab_cop vector is then used to compute the body forces and moments by a call to the CSC sum_body_forces_and_moments_about_ac.

```
vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
```

See APPENDIX B for a complete source code listing.

3.1.23 VSTAB_LIFT_COEFFICIENT_1

VSTAB_LIFT_COEFFICIENT_1 is a constant defining the second coefficient of the vertical stabilator coefficient polynomial.

3.1.23.1 Initialization

The constant VSTAB_LIFT_COEFFICIENT_1 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define VSTAB_LIFT_COEFFICIENT_1          aero_data[22]
```

3.1.23.2 Usage

During real-time execution, this variable is not recomputed.

3.1.23.2.1 Algorithm

VSTAB_LIFT_COEFFICIENT_1 is used to compute the unit lift coefficient of the vertical stabilator in the CSU vstab_lift_coefficient.

```
if (abs(yaw) > VSTAB_STALL_SSA)
    yawval = sign(yawval) * VSTAB_STALL_SSA;
else
    yawval = yaw;

return (VSTAB_LIFT_COEFFICIENT_1 * yawval);
```

The vstab_lift_coefficient is used to compute the total vertical stabilator lift by a call to the CSU compute_lift-drag_coefficients.

```
lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
```

See APPENDIX B for a complete source code listing.

3.1.24 VSTAB_STALL_SSA

VSTAB_STALL_SSA is a constant defining the stall angle of the vertical stabilator.

3.1.24.1 Initialization

The constant VSTAB_STALL_SSA is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  VSTAB_STALL_SSA            (deg_to_rad(aero_data[23]))
```

3.1.24.2 Usage

During real-time execution, this variable is not recomputed.

3.1.24.2.1 Algorithm

VSTAB_STALL_SSA is used to compute the unit lift coefficient of the vertical stabilator in the CSU vstab_lift_coefficient.

```
if (abs(yaw) > VSTAB_STALL_SSA)
    yawval = sign(yawval) * VSTAB_STALL_SSA;
else
    yawval = yaw;

return (VSTAB_LIFT_COEFFICIENT_1 * yawval);
```

The vstab_lift_coefficient is used to compute the total vertical stabilator lift by a call to the CSU compute_lift_drag_coefficients.

```
lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
```

See APPENDIX B for a complete source code listing.

3.1.25 MAIN_ROTOR_COP_AC_X

MAIN_ROTOR_COP_AC_X is a constant defining the location in the x-axis of the center of pressure for the main rotor.

3.1.25.1 Initialization

The constant MAIN_ROTOR_COP_AC_X is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_COP_AC_X                aero_data[24]
```

3.1.25.2 Usage

During real-time execution, this variable is not recomputed.

3.1.25.2.1 Algorithm

MAIN_ROTOR_COP_AC_X is used to initialize the location in the x-axis of the center of pressure for the main rotor in the CSU aerodyn_init.

```
loc_ac_main_rotor_cop[X] = MAIN_ROTOR_COP_AC_X;
```

See APPENDIX B for a complete source code listing.

3.1.26 MAIN_ROTOR_COP_AC_Y

MAIN_ROTOR_COP_AC_Y is a constant defining the location in the y-axis of the center of pressure for the main rotor.

3.1.26.1 Initialization

The constant MAIN_ROTOR_COP_AC_Y is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_COP_AC_Y            aero_data[25]
```

3.1.26.2 Usage

During real-time execution, this variable is not recomputed.

3.1.26.2.1 Algorithm

MAIN_ROTOR_COP_AC_Y is used to initialize the location in the y-axis of the center of pressure for the main rotor in the CSU aerodyn_init.

```
loc_ac_main_rotor_cop[Y] = MAIN_ROTOR_COP_AC_Y;
```

See APPENDIX B for a complete source code listing.

3.1.27 MAIN_ROTOR_COP_AC_Z

MAIN_ROTOR_COP_AC_Z is a constant defining the location in the z-axis of the center of pressure for the main rotor.

3.1.27.1 Initialization

The constant MAIN_ROTOR_COP_AC_Z is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU

aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MAIN_ROTOR_COP_AC_Z                aero_data[26]
```

3.1.27.2 Usage

During real-time execution, this variable is not recomputed.

3.1.27.2.1 Algorithm

MAIN_ROTOR_COP_AC_Z is used to initialize the location in the z-axis of the center of pressure for the main rotor in the CSU aerodyn_init.

```
loc_ac_main_rotor_cop[Z] = MAIN_ROTOR_COP_AC_Z;
```

See APPENDIX B for a complete source code listing.

3.1.28 MAIN_ROTOR_MAX_THRUST

MAIN_ROTOR_MAX_THRUST is a constant defining the maximum thrust of the main rotor at 100 per cent rpm.

3.1.28.1 Initialization

The constant MAIN_ROTOR_MAX_THRUST is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MAIN_ROTOR_MAX_THRUST              aero_data[27]
```

3.1.28.2 Usage

During real-time execution, this variable is not recomputed.

3.1.28.2.1 Algorithm

MAIN_ROTOR_MAX_THRUST is used to compute main_rotor_thrust in the CSU compute_rotor_forces_and_moments.

```
main_rotor_thrust = powertrain_percent_shaft_speed *
                    controller_collective *
                    MAIN_ROTOR_MAX_THRUST;
```

See APPENDIX B for a complete source code listing.

3.1.29 MAIN_ROTOR_MAST_TILT

MAIN_ROTOR_MAST_TILT is a constant defining the angle of tilt of the main rotor mast.

3,1.29.1 Initialization

The constant MAIN_ROTOR_MAST_TILT is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MAIN_ROTOR_MAST_TILT          (deg_to_rad(aero_data[28]))
```

3.1.29.2 Usage

During real-time execution, this variable is not recomputed.

3.1.29.2.1 Algorithm

MAIN_ROTOR_MAST_TILT is used to compute sine and cosine of the angle of main rotor mast tilt in the CSU aerodyn_init.

```
MAIN_ROTOR_MAST_TILT_SIN = sin(MAIN_ROTOR_MAST_TILT);
MAIN_ROTOR_MAST_TILT_COS = cos(MAIN_ROTOR_MAST_TILT);
```

These values are used to compute the forces generated by the main rotor on the body by a call to the CSU compute_rotor_forces_and_moments.

```
force_body_main_rotor[Y] = main_rotor_thrust *
                    MAIN_ROTOR_MAST_TILT_SIN;

force_body_main_rotor[Z] = main_rotor_thrust *
                    MAIN_ROTOR_MAST_TILT_COS;
```

See APPENDIX B for a complete source code listing.

3.1.30 MAIN_ROTOR_MAX_LOAD_TORQUE

MAIN_ROTOR_MAX_LOAD_TORQUE is a constant defining the maximum load torque of the main rotor.

3.1.30.1 Initialization

The constant MAIN_ROTOR_MAX_LOAD_TORQUE is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_MAX_LOAD_TORQUE          aero_data[29]
```

3.1.30.2 Usage

During real-time execution, this variable is not recomputed.

3.1.30.2.1 Algorithm

MAIN_ROTOR_MAX_LOAD_TORQUE is used to compute the load torque of the main rotor by a call to the CSU compute_rotor_loads.

```
main_rotor_load_torque = controller_collective *
                         MAIN_ROTOR_MAX_LOAD_TORQUE;
```

The main_rotor_load_torque is used to compute the engine torque by a call to the CSU compute_engine_torque.

```
engine_simul(main_rotor_load_torque,
             tail_rotor_load_torque, altitude);
```

See APPENDIX B for a complete source code listing.

3.1.31 MAIN_ROTOR_MAX_PITCH_MOMENT

MAIN_ROTOR_MAX_PITCH_MOMENT is a constant defining the maximum pitching moment of the main rotor.

3.1.31.1 Initialization

The constant MAIN_ROTOR_MAX_PITCH_MOMENT is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_MAX_PITCH_MOMENT        aero_data[30]
```

3.1.31.2 Usage

During real-time execution, this variable is not recomputed.

3.1.31.2.1 Algorithm

MAIN_ROTOR_MAX_PITCH_MOMENT is used to compute the pitching moment on the body generated by the main rotor by a call to the CSU compute_rotor_forces_and_moments.

```
moment_body_main_rotor[X] =
   - controller_cyclic_pitch * MAIN_ROTOR_MAX_PITCH_MOMENT;
```

The components are summed for the total moment on the body by a call to the CSU sum_body_forces_and_moments_about_ac.

```
vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

3.1.32 MAIN_ROTOR_MAX_ROLL_MOMENT

MAIN_ROTOR_MAX_ROLL_MOMENT is a constant defining the maximum rolling moment of the main rotor.

3.1.32.1 Initialization

The constant MAIN_ROTOR_MAX_ROLL_MOMENT is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_MAX_ROLL_MOMENT        aero_data[31]
```

3.1.32.2 Usage

During real-time execution, this variable is not recomputed.

3.1.32.2.1 Algorithm

MAIN_ROTOR_MAX_ROLL_MOMENT is used to compute the rolling moment on the body generated by the main rotor by a call to the CSU compute_rotor_forces_and_moments.

```
moment_body_main_rotor[Y] =
    controller_cyclic_roll * MAIN_ROTOR_MAX_ROLL_MOMENT;
```

The components are summed for the total moment on the body by a call to the CSU sum_body_forces_and_moments_about_ac.

```
vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

3.1.33 MAIN_ROTOR_TORQUE_COUPLING_GAIN

MAIN_ROTOR_TORQUE_COUPLING_GAIN is a constant defining the torque moment generated by the main rotor.

3.1.33.1 Initialization

The constant MAIN_ROTOR_TORQUE_COUPLING_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_TORQUE_COUPLING_GAIN    aero_data[32]
```

3.1.33.2 Usage

During real-time execution, this variable is not recomputed.

3.1.33.2.1 Algorithm

MAIN_ROTOR_TORQUE_COUPLING_GAIN is used to compute the torque moment on the body generated by the main rotor by a call to the CSU compute_rotor_forces_and_moments.

```
moment_body_main_rotor[Z] =
        - main_rotor_load_torque *
        MAIN_ROTOR_TORQUE_COUPLING_GAIN;
```

The components are summed for the total moment on the body by a call to the CSU sum_body_forces_and_moments_about_ac.

```
vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

3.1.34 MAIN_ROTOR_GROUND_EFFECT_FACTOR

MAIN_ROTOR_GROUND_EFFECT_FACTOR is a constant defining the ground effect on the main rotor.

3.1.34.1 Initialization

The constant MAIN_ROTOR_GROUND_EFFECT_FACTOR is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_GROUND_EFFECT_FACTOR    aero_data[33]
```

3.1.34.2 Usage

During real-time execution, this variable is not recomputed.

3.1.34.2.1 Algorithm

MAIN_ROTOR_GROUND_EFFECT_FACTOR is used to compute the ground effect force generated during proximity of the rotor and the ground by a call to the CSC interact_with_ground.

```
force_ground_effect[Z] = main_rotor_thrust
                     * MAIN_ROTOR_GROUND_EFFECT_FACTOR
                     / (cig_altitude_above_gnd() + 1.0);
```

See APPENDIX B for a complete source code listing.

3.1.35 TAIL_ROTOR_COP_AC_X

TAIL_ROTOR_COP_AC_X is a constant defining the location in the x-axis of the center of pressure of the tail rotor for the vehicle.

3.1.35.1 Initialization

The constant TAIL_ROTOR_COP_AC_X is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define TAIL_ROTOR_COP_AC_X                aero_data[34]
```

3.1.35.2 Usage

During real-time execution, this variable is not recomputed.

3.1.35.2.1 Algorithm

TAIL_ROTOR_COP_AC_X is used to initialize the location in the x-axis of the center of pressure for the tail rotor by a call to the CSU aerodyn_init.

```
loc_ac_tail_rotor_cop[X] = TAIL_ROTOR_COP_AC_X;
```

The loc_ac_tail_rotor_cop vector is then used to compute and sum the body forces and moments by a call to the CSU sum_body_forces_and_moments_ac.

```
vec_cross_prod(loc_ac_tail_rotor_cop, force_body_tail_rotor,
                moment_body_tail_rotor);
vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                moment_body_virtual_wing);
vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
vec_cross_prod(loc_ac_cg, gravity_force_body, moment_body_cg);

vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

3.1.36 TAIL_ROTOR_COP_AC_Y

TAIL_ROTOR_COP_AC_Y is a constant defining the location in the y-axis of the center of pressure of the tail rotor for the vehicle.

3.1.36.1 Initialization

The constant TAIL_ROTOR_COP_AC_Y is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define TAIL_ROTOR_COP_AC_Y          aero_data[35]
```

3.1.36.2 Usage

During real-time execution, this variable is not recomputed.

3.1.36.2.1 Algorithm

TAIL_ROTOR_COP_AC_Y is used to initialize the location in the y-axis of the center of pressure for the tail rotor by a call to the CSU aerodyn_init.

```
loc_ac_tail_rotor_cop[Y] = TAIL_ROTOR_COP_AC_Y;
```

The loc_ac_tail_rotor_cop vector is then used to compute and sum the body forces and moments by a call to the CSU sum_body_forces_and_moments_ac.

```
vec_cross_prod(loc_ac_tail_rotor_cop, force_body_tail_rotor,
                moment_body_tail_rotor);
vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                moment_body_virtual_wing);
vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
vec_cross_prod(loc_ac_cg, gravity_force_body, moment_body_cg);

vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

3.1.37 TAIL_ROTOR_COP_AC_Z

TAIL_ROTOR_COP_AC_Z is a constant defining the location in the z-axis of the center of pressure of the tail rotor for the vehicle.

3.1.37.1 Initialization

The constant TAIL_ROTOR_COP_AC_Z is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define TAIL_ROTOR_COP_AC_Z              aero_data[36]
```

3.1.37.2 Usage

During real-time execution, this variable is not recomputed.

3.1.37.2.1 Algorithm

TAIL_ROTOR_COP_AC_Z is used to initialize the location in the z-axis of the center of pressure for the tail rotor by a call to the CSU aerodyn_init.

```
loc_ac_tail_rotor_cop[Z] = TAIL_ROTOR_COP_AC_Z;
```

The loc_ac_tail_rotor_cop vector is then used to compute and sum the body forces and moments by a call to the CSU sum_body_forces_and_moments_ac.

```
vec_cross_prod(loc_ac_tail_rotor_cop, force_body_tail_rotor,
                         moment_body_tail_rotor);
vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                         moment_body_virtual_wing);
vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
vec_cross_prod(loc_ac_cg, gravity_force_body, moment_body_cg);

vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

3.1.38 TAIL_ROTOR_MAX_THRUST

TAIL_ROTOR_MAX_THRUST is a constant defining the maximum thrust of the tail rotor.

3.1.38.1 Initialization

The constant TAIL_ROTOR_MAX_THRUST is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU

aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  TAIL_ROTOR_MAX_THRUST                    aero_data[37]
```

3.1.38.2 Usage

During real-time execution, this variable is not recomputed.

3.1.38.2.1 Algorithm

TAIL_ROTOR_MAX_THRUST is used to compute the tail rotor thrust and its force on the body of the vehicle by a call to the CSU compute_rotor_forces_and_moments.

```
  tail_rotor_thrust = powertrain_percent_shaft_speed *
            controller_tail_rotor  *  TAIL_ROTOR_MAX_THRUST;
. . . .
  force_body_tail_rotor[X] = tail_rotor_thrust;
```

See APPENDIX B for a complete source code listing.

3.1.39 TAIL_ROTOR_MAX_LOAD_TORQUE

TAIL_ROTOR_MAX_LOAD_TORQUE is a constant defining the maximum load torque of the tail rotor at 100 percent rpm.

3.1.39.1 Initialization

The constant TAIL_ROTOR_MAX_LOAD_TORQUE is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  TAIL_ROTOR_MAX_LOAD_TORQUE               aero_data[38]
```

3.1.39.2 Usage

During real-time execution, this variable is not recomputed.

3.1.39.2.1 Algorithm

TAIL_ROTOR_MAX_LOAD_TORQUE is used to load torque for the tail rotor by a call to the CSU compute_rotor_loads.

```
tail_rotor_load_torque = abs (controller_tail_rotor) *
                         TAIL_ROTOR_MAX_LOAD_TORQUE;
```

The tail_rotor_load_torque is used to compute the engine torque by a call to the CSU compute_engine_torque.

```
engine_simul(main_rotor_load_torque,
             tail_rotor_load_torque, altitude);
```

See APPENDIX B for a complete source code listing.

3.1.40 P_DRAG_COEFF_CONST

P_DRAG_COEFF_CONST is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

3.1.40.1 Initialization

The constant P_DRAG_COEFF_CONST is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define P_DRAG_COEFF_CONST                aero_data[39]
```

3.1.40.2 Usage

During real-time execution, this variable is not recomputed.

3.1.40.2.1 Algorithm

P_DRAG_COEFF_CONST is used to compute the parasitic drag profile coefficient from a cubic function by a call to the CSU find_cubic_function.

The p_drag_fit_coeff vector is initialized to zero, then computed. If an error, especially the result is 0.0, an error statement is generated. The p_drag_fit_coeff vector is computed by a call to the CSU aerodyn_init.

```
for (i=0; i<9; i++)                    /* Set parasite drag profile */
{
    p_drag_fit_coeff[i] = 0.0;
}

 if (find_cubic_func (0.0, P_DRAG_COEFF_CONST,
                 P_DRAG_TAS_BREAK, P_DRAG_COEFF_BREAK,
                 P_DRAG_TAS_MAX, P_DRAG_COEFF_MAX,
                 0.5, p_drag_fit_coeff) != TRUE)
{
    fprintf (stderr, "AERODYN: Error - unable to fit p_drag function\n");
}
```

The p_drag_fit_coeff vector is then used to compute the total incompressible_drag_coefficient by a call to the CSU compute_lift_drag_coefficients.

```
   lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
/* Computing virtual wing coefficient as independent of AOA */
   lift_coefficient_virtual_wing = LIFT_COEFF_VIRTUAL_WING;
/*                   virtual_wing_lift_coefficient (angle_of_attack); */

  parasite_drag_coefficient = cubic_func (true_airspeed, p_drag_fit_coeff);

  if (true_airspeed > 0.0 && angle_of_attack > 0.0)   /* speed brake */
 {
     multiplier = 5.0 * true_airspeed * sin(angle_of_attack);
     if (multiplier > 1.0)
         parasite_drag_coefficient *= multiplier;
 }

  oswald_efficiency_factor = OSWALD_EFFIC_FACTOR;

  induced_drag_coefficient = INDUCED_DRAG_COEFF;

  total_incompressible_drag_coefficient = parasite_drag_coefficient +
                                  induced_drag_coefficient;
```

The total drag is computed by a call to the CSU compute_lift_drag_forces.

```
total_drag = total_incompressible_drag_coefficient * dynamic_pressure *
                    TOTAL_WETTED_SURFACE_AREA;
```

See APPENDIX B for a complete source code listing.

3.1.41 P_DRAG_TAS_BREAK

P_DRAG_TAS_BREAK is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

3.1.41.1 Initialization

The constant P_DRAG_TAS_BREAK is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  P_DRAG_TAS_BREAK                    aero_data[40]
```

3.1.41.2 Usage

During real-time execution, this variable is not recomputed.

3.1.41.2.1 Algorithm

See paragraph 3.1.40.

P_DRAG_TAS_BREAK is used to control computation of the y-axis element of the drag force by a call to the CSU transform_lift_drag_forces_to_body_coordinates.

```
virtual_wing_force[Z]  = lift_virtual_wing;        /* [H, D, L] */
vstab_force[X] =  lift_vstab;
drag_force[Y] = -total_drag;

if (true_airspeed < P_DRAG_TAS_BREAK)                   /* jwc 8/90 */
    drag_force[Y] -= sin(pitch) * 50000;

        vec_mat_mul  (virtual_wing_force,  velocity_to_body,
lift_body_virtual_wing);
vec_mat_mul (vstab_force, velocity_to_body, lift_body_vstab);
vec_mat_mul (drag_force, velocity_to_body, drag_body);
```

See APPENDIX B for a complete source code listing.

3.1.42 P_DRAG_COEFF_BREAK

P_DRAG_COEFF_BREAK is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

3.1.42.1 Initialization

The constant P_DRAG_COEFF_BREAK is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define P_DRAG_COEFF_BREAK                  aero_data[41]
```

3.1.42.2 Usage

During real-time execution, this variable is not recomputed.

3.1.42.2.1 Algorithm

See paragraph 3.1.40.

See APPENDIX B for a complete source code listing.

3.1.43 P_DRAG_TAS_MAX

P_DRAG_TAS_MAX is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

3.1.43.1 Initialization

The constant P_DRAG_TAS_MAX is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  P_DRAG_TAS_MAX                    aero_data[42]
```

3.1.43.2 Usage

During real-time execution, this variable is not recomputed.

3.1.43.2.1 Algorithm

See paragraph 3.1.40.

See APPENDIX B for a complete source code listing.

3.1.44 P_DRAG_COEFF_MAX

P_DRAG_COEFF_MAX is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

3.1.44.1 Initialization

The constant P_DRAG_COEFF_MAX is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  P_DRAG_COEFF_MAX                  aero_data[43]
```

3.1.44.2 Usage

During real-time execution, this variable is not recomputed.

3.1.44.2.1 Algorithm

See paragraph 3.1.40.

See APPENDIX B for a complete source code listing.

3.1.45 TOTAL_WETTED_SURFACE_AREA

TOTAL_WETTED_SURFACE_AREA is a constant defining the total wetted surface area of the vehicle.

3.1.45.1 Initialization

The constant TOTAL_WETTED_SURFACE_AREA is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  TOTAL_WETTED_SURFACE_AREA              aero_data[44]
```

3.1.45.2 Usage

During real-time execution, this variable is not recomputed.

3.1.45.2.1 Algorithm

TOTAL_WETTED_SURFACE_AREA is used to compute the total drag by a call to the CSU compute_lift_drag_forces.

```
total_drag = total_incompressible_drag_coefficient * dynamic_pressure *
                    TOTAL_WETTED_SURFACE_AREA;
```

See APPENDIX B for a complete source code listing.

3.1.46 MAX_ATT_CTL_ANGLE_STOP

MAX_ATT_CTL_ANGLE_STOP is a constant defining the maximum attitude control angle stop.

3.1.46.1 Initialization

The constant MAX_ATT_CTL_ANGLE_STOP is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MAX_ATT_CTL_ANGLE_STOP            aero_data[45]
```

3.1.46.2 Usage

During real-time execution, this variable is not recomputed.

3.1.46.2.1 Algorithm

MAX_ATT_CTL_ANGLE_STOP is used to limit the maximum attitude control angle for the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
#if ATT_DAMPING_MODE_SIMPLE
	if (true_airspeed > HOVER_SLOW_LIMIT )
		MAX_ATT_CTL_ANGLE =
			log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
	else if (true_airspeed < -HOVER_SLOW_LIMIT )
		MAX_ATT_CTL_ANGLE =
			log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
	else
		MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_STOP ;

	MAX_ATT_CTL_ANGLE = deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif
```

See APPENDIX B for a complete source code listing.

3.1.47 MAX_ATT_DAMPING_FACTOR

MAX_ATT_DAMPING_FACTOR is a constant defining the

3.1.47.1 Initialization

The constant MAX_ATT_DAMPING_FACTOR is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_ATT_DAMPING_FACTOR          aero_data[46]
```

3.1.47.2 Usage

During real-time execution, this variable is not recomputed.

3.1.47.2.1 Algorithm

MAX_ATT_DAMPING_FACTOR is used to compute the maximum attitude control angle for the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
#if ATT_DAMPING_MODE_SIMPLE
        if (true_airspeed > HOVER_SLOW_LIMIT )
                MAX_ATT_CTL_ANGLE =
                        log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
        else if (true_airspeed < -HOVER_SLOW_LIMIT )
                MAX_ATT_CTL_ANGLE =
                        log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
        else
                MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_STOP ;

        MAX_ATT_CTL_ANGLE = deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif
```

See APPENDIX B for a complete source code listing.

3.1.48 HOVER_SLOW_LIMIT

HOVER_SLOW_LIMIT is a constant defining the slow limit speed in hover.

3.1.48.1 Initialization

The constant HOVER_SLOW_LIMIT is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_SLOW_LIMIT                    aero_data[47]
```

3.1.48.2 Usage

During real-time execution, this variable is not recomputed.

3.1.48.2.1 Algorithm

HOVER_SLOW_LIMIT is used to control computation of the maximum attitude control angle for the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
#if ATT_DAMPING_MODE_SIMPLE
            if  (true_airspeed < HOVER_SLOW_LIMIT)
              {
               if (true_airspeed > -HOVER_SLOW_LIMIT)
                   MAX_ATT_CTL_ANGLE =
                          MAX_ATT_CTL_ANGLE_SLOW  ;
               else if (true_airspeed > -HOVER_MED_LIMIT)
                   MAX_ATT_CTL_ANGLE =
                          MAX_ATT_CTL_ANGLE_MED;
              else
                   MAX_ATT_CTL_ANGLE =
                          MAX_ATT_CTL_ANGLE_NORM  ;
              }
            else if (true_airspeed < HOVER_MED_LIMIT)
                   MAX_ATT_CTL_ANGLE =
                          MAX_ATT_CTL_ANGLE_MED  ;
            else
                   MAX_ATT_CTL_ANGLE =
                           MAX_ATT_CTL_ANGLE_NORM  ;
#endif
```

```
#if ATT_DAMPING_MODE_SIMPLE
        if (true_airspeed > HOVER_SLOW_LIMIT )
                MAX_ATT_CTL_ANGLE =
                        log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
        else if (true_airspeed < -HOVER_SLOW_LIMIT )
                MAX_ATT_CTL_ANGLE =
                        log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
        else
                MAX_ATT_CTL_ANGLE =
                        MAX_ATT_CTL_ANGLE_STOP ;
                MAX_ATT_CTL_ANGLE =
                        deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif
```

See APPENDIX B for a complete source code listing.

3.1.49 HOVER_AUG_PITCH_RESET_VALUE

HOVER_AUG_PITCH_RESET_VALUE is a constant defining the value for the hover augmentation pitch integrator is reset when hover hold is turned on.

3.1.49.1 Initialization

The constant HOVER_AUG_PITCH_RESET_VALUE is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_AUG_PITCH_RESET_VALUE          aero_data[48]
```

3.1.49.2 Usage

During real-time execution, this variable is not recomputed.

3.1.49.2.1 Algorithm

HOVER_AUG_PITCH_RESET_VALUE is used to compute the hover augmentation pitch integrator by a call to the CSU compute_stab_augmentation_gains.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
            {
            hover_hold_turned_on = TRUE ;


....
            /* You should already be "hovering" (airspeed < 10 knots)
              for hover hold to show little visible swaying. */

            hover_aug_roll_integrator = 0.0 ;
            hover_aug_pitch_integrator =
                  HOVER_AUG_PITCH_RESET_VALUE ;
....

      hover_aug_pitch_integrator +=
                  HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
      hover_aug_pitch_integrator =
                  limiter(-0.2,hover_aug_pitch_integrator,0.2);
      hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                  velocity_vector[Y]
                  + hover_aug_pitch_integrator;
      hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_pitch_angle,
                              MAX_ATT_CTL_ANGLE);
....
  }
  else
  {
....
#ifdef notdef
      hover_aug_roll_integrator = 0.0;        /* added 8/31/89 (jwc) */
      hover_aug_pitch_integrator = 0.0;
#endif
  }
```

See APPENDIX B for a complete source code listing.

3.1.50 MAX_ATT_CTL_ANGLE_NORM

MAX_ATT_CTL_ANGLE_NORM is a constant defining the normal setting for the maximum attitude control angle.

3.1.50.1 Initialization

The constant MAX_ATT_CTL_ANGLE_NORM is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MAX_ATT_CTL_ANGLE_NORM     (deg_to_rad (aero_data[49]))
```

3.1.50.2 Usage

During real-time execution, this variable is not recomputed.

3.1.50.2.1 Algorithm

MAX_ATT_CTL_ANGLE_NORM is used to initialize the maximum attitude control angle to the normal setting for the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
        if ( !hover_hold_turned_on )
                {
                hover_hold_turned_on = TRUE ;

. . . .

#if  ATT_DAMPING_MODE_SIMPLE
                if  (true_airspeed < HOVER_SLOW_LIMIT)
                   {
                    if (true_airspeed > -HOVER_SLOW_LIMIT)
                        MAX_ATT_CTL_ANGLE  =
                                        MAX_ATT_CTL_ANGLE_SLOW  ;
                    else if (true_airspeed > -HOVER_MED_LIMIT)
                        MAX_ATT_CTL_ANGLE  =
                                        MAX_ATT_CTL_ANGLE_MED;
                   else
                        MAX_ATT_CTL_ANGLE  =
                                        MAX_ATT_CTL_ANGLE_NORM  ;
                   }
                else if (true_airspeed < HOVER_MED_LIMIT)
                        MAX_ATT_CTL_ANGLE  =
                                        MAX_ATT_CTL_ANGLE_MED  ;
```

```
                else
                        MAX_ATT_CTL_ANGLE =
                                MAX_ATT_CTL_ANGLE_NORM ;
#endif
                }
```

See APPENDIX B for a complete source code listing.

3.1.51 MAX_ATT_CTL_ANGLE_MED

MAX_ATT_CTL_ANGLE_MED is a constant defining the medium setting for the maximum attitude control angle.

3.1.51.1 Initialization

The constant MAX_ATT_CTL_ANGLE_MED is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  MAX_ATT_CTL_ANGLE_MED        (deg_to_rad (aero_data[50]))
```

3.1.51.2 Usage

During real-time execution, this variable is not recomputed.

3.1.51.2.1 Algorithm

MAX_ATT_CTL_ANGLE_MED is used to initialize the maximum attitude control angle to the normal setting for the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
        if ( !hover_hold_turned_on )
                {
                hover_hold_turned_on = TRUE ;

. . . .

#if  ATT_DAMPING_MODE_SIMPLE
                if  (true_airspeed < HOVER_SLOW_LIMIT)
                   {
                   if (true_airspeed > -HOVER_SLOW_LIMIT)
```

```
                MAX_ATT_CTL_ANGLE =
                            MAX_ATT_CTL_ANGLE_SLOW ;
            else if (true_airspeed > -HOVER_MED_LIMIT)
                MAX_ATT_CTL_ANGLE =
                            MAX_ATT_CTL_ANGLE_MED;
            else
                MAX_ATT_CTL_ANGLE =
                            MAX_ATT_CTL_ANGLE_NORM ;
            }
        else if (true_airspeed < HOVER_MED_LIMIT)
                MAX_ATT_CTL_ANGLE =
                            MAX_ATT_CTL_ANGLE_MED ;
        else
                MAX_ATT_CTL_ANGLE =
                            MAX_ATT_CTL_ANGLE_NORM ;
#endif
        }
```

See APPENDIX B for a complete source code listing.

3.1.52 MAX_ATT_CTL_ANGLE_SLOW

MAX_ATT_CTL_ANGLE_SLOW is a constant defining the slow setting for the maximum attitude control angle.

3.1.52.1 Initialization

The constant MAX_ATT_CTL_ANGLE_SLOW is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_ATT_CTL_ANGLE_SLOW      (deg_to_rad (aero_data[51]))
```

3.1.52.2 Usage

During real-time execution, this variable is not recomputed.

3.1.52.2.1 Algorithm

MAX_ATT_CTL_ANGLE_SLOW is used to initialize the maximum attitude control angle to the slow setting for the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
        if ( !hover_hold_turned_on )
                {
                hover_hold_turned_on = TRUE ;

....

#if ATT_DAMPING_MODE_SIMPLE
                if  (true_airspeed < HOVER_SLOW_LIMIT)
                  {
                   if (true_airspeed > -HOVER_SLOW_LIMIT)
                       MAX_ATT_CTL_ANGLE =
                                       MAX_ATT_CTL_ANGLE_SLOW  ;
                   else if (true_airspeed > -HOVER_MED_LIMIT)
                       MAX_ATT_CTL_ANGLE =
                                       MAX_ATT_CTL_ANGLE_MED;
                  else
                       MAX_ATT_CTL_ANGLE =
                                       MAX_ATT_CTL_ANGLE_NORM  ;
                  }
                else if (true_airspeed < HOVER_MED_LIMIT)
                       MAX_ATT_CTL_ANGLE =
                                       MAX_ATT_CTL_ANGLE_MED  ;
                else
                       MAX_ATT_CTL_ANGLE =
                                       MAX_ATT_CTL_ANGLE_NORM  ;
#endif
                }
```

See APPENDIX B for a complete source code listing.

3.1.53 HOVER_MED_LIMIT

HOVER_MED_LIMIT is a constant defining the medium speed limit for hover.

3.1.53.1 Initialization

The constant HOVER_MED_LIMIT is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_MED_LIMIT                    aero_data[52]
```

3.1.53.2 Usage

During real-time execution, this variable is not recomputed.

3.1.53.2.1 Algorithm

HOVER_MED_LIMIT is used to control the computation of the maximum attitude control angle in the simple flight mode by a call to the CSU compute_stab_augmentation_gains.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
            {
....

#if ATT_DAMPING_MODE_SIMPLE
            if  (true_airspeed < HOVER_SLOW_LIMIT)
              {
               if (true_airspeed > -HOVER_SLOW_LIMIT)
                   MAX_ATT_CTL_ANGLE =
                           MAX_ATT_CTL_ANGLE_SLOW  ;
               else if (true_airspeed > -HOVER_MED_LIMIT)
                   MAX_ATT_CTL_ANGLE =
                           MAX_ATT_CTL_ANGLE_MED;
              else
                   MAX_ATT_CTL_ANGLE =
                           MAX_ATT_CTL_ANGLE_NORM  ;
              }
            else if (true_airspeed < HOVER_MED_LIMIT)
                   MAX_ATT_CTL_ANGLE =
                           MAX_ATT_CTL_ANGLE_MED  ;
            else
                   MAX_ATT_CTL_ANGLE =
                   MAX_ATT_CTL_ANGLE_NORM  ;
#endif
            }
....
```

See APPENDIX B for a complete source code listing.

3.1.54 ATT_CTL_PITCH_P_GAIN

ATT_CTL_PITCH_P_GAIN is a constant defining the slope for the attitude control pitch command equation.

3.1.54.1 Initialization

The constant ATT_CTL_PITCH_P_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  ATT_CTL_PITCH_P_GAIN                aero_data[53]
```

3.1.54.2 Usage

During real-time execution, this variable is not recomputed.

3.1.54.2.1 Algorithm

ATT_CTL_PITCH_P_GAIN is used to compute the attitude control pitch command by a call to the CSU set_pitch_attitude.

```
attitude_control_pitch_integrator +=
        ATT_CTL_PITCH_I_GAIN * (pitch - angle);
attitude_control_pitch_integrator =
        limiter (-0.1, attitude_control_pitch_integrator, 0.1);
attitude_control_pitch_command = ATT_CTL_PITCH_P_GAIN *
            (pitch - angle);
attitude_control_pitch_command += attitude_control_pitch_integrator;
attitude_control_pitch_command = limiter (
            -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
            attitude_control_pitch_command,
            MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_pitch_command);
```

See APPENDIX B for a complete source code listing.

3.1.55 ATT_CTL_PITCH_I_GAIN

ATT_CTL_PITCH_I_GAIN is a constant defining the slope for the attitude control pitch integrator equation.

3.1.55.1 Initialization

The constant ATT_CTL_PITCH_I_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define ATT_CTL_PITCH_I_GAIN                aero_data[54]
```

3.1.55.2 Usage

During real-time execution, this variable is not recomputed.

3.1.55.2.1 Algorithm

ATT_CTL_PITCH_I_GAIN is used to compute the attitude control pitch integrator by a call to the CSU set_pitch_attitude. The integrator is used to compute the attitude control pitch command.

```
attitude_control_pitch_integrator +=
        ATT_CTL_PITCH_I_GAIN * (pitch - angle);
attitude_control_pitch_integrator =
        limiter (-0.1, attitude_control_pitch_integrator, 0.1);
attitude_control_pitch_command = ATT_CTL_PITCH_P_GAIN *
                (pitch - angle);
attitude_control_pitch_command += attitude_control_pitch_integrator;
attitude_control_pitch_command = limiter (
                -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
                attitude_control_pitch_command,
                MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_pitch_command);
```

See APPENDIX B for a complete source code listing.

3.1.56 ATT_CTL_ROLL_P_GAIN

ATT_CTL_ROLL_P_GAIN is a constant defining the slope for the attitude control roll command equation.

3.1.56.1 Initialization

The constant ATT_CTL_ROLL_P_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define ATT_CTL_ROLL_P_GAIN            aero_data[55]
```

3.1.56.2 Usage

During real-time execution, this variable is not recomputed.

3.1.56.2.1 Algorithm

ATT_CTL_ROLL_P_GAIN is used to compute the attitude control roll command by a call to the CSU set_roll_attitude.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
        (roll - angle);
/**** These used to be attitude_control_pitch_integrator instead of
    attitude_control_roll_integrator.      PJM   11-1-89
attitude_control_pitch_integrator =
        limiter (-0.1, attitude_control_pitch_integrator, 0.1);
******/
attitude_control_roll_integrator =
        limiter (-0.1, attitude_control_roll_integrator, 0.1);
attitude_control_roll_command = ATT_CTL_ROLL_P_GAIN *
        (roll - angle);
attitude_control_roll_command += attitude_control_roll_integrator;
attitude_control_roll_command =  limiter (
        -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
        attitude_control_roll_command,
        MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_roll_command);
```

See APPENDIX B for a complete source code listing.

3.1.57 ATT_CTL_ROLL_I_GAIN

ATT_CTL_ROLL_I_GAIN is a constant defining the slope for the attitude control roll integrator equation.

3.1.57.1 Initialization

The constant ATT_CTL_ROLL_I_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  ATT_CTL_ROLL_I_GAIN                    aero_data[56]
```

3.1.57.2 Usage

During real-time execution, this variable is not recomputed.

3.1.57.2.1 Algorithm

ATT_CTL_ROLL_I_GAIN is used to compute the attitude control roll integrator by a call to the CSU set_roll_attitude. The integrator is used to compute the attitude control roll command.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
          (roll - angle);
/**** These used to be attitude_control_pitch_integrator instead of
     attitude_control_roll_integrator.       PJM   11-1-89
attitude_control_pitch_integrator =
          limiter (-0.1, attitude_control_pitch_integrator, 0.1);
******/
attitude_control_roll_integrator =
          limiter (-0.1, attitude_control_roll_integrator, 0.1);
attitude_control_roll_command = ATT_CTL_ROLL_P_GAIN *
          (roll - angle);
attitude_control_roll_command += attitude_control_roll_integrator;
attitude_control_roll_command =  limiter (
          -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
          attitude_control_roll_command,
          MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_roll_command);
```

See APPENDIX B for a complete source code listing.

3.1.58 HOVER_AUG_ROLL_P_GAIN

HOVER_AUG_ROLL_P_GAIN is a constant defining the slope for the hover augmentation roll angle equation.

3.1.58.1 Initialization

The constant HOVER_AUG_ROLL_P_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_ROLL_P_GAIN                    aero_data[57]
```

3.1.58.2 Usage

During real-time execution, this variable is not recomputed.

3.1.58.2.1 Algorithm

HOVER_AUG_ROLL_P_GAIN is used to compute the hover augmentation roll angle by a call to the CSU compute_stab_augmentation_gains.

```
 if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
           {
           hover_hold_turned_on = TRUE ;
....
           /* You should already be "hovering" (airspeed < 10 knots)
              for hover hold to show little visible swaying. */

           hover_aug_roll_integrator = 0.0 ;
....
           }
....
    hover_aug_roll_integrator +=
                  HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
    hover_aug_roll_integrator =
                  limiter(-0.2,hover_aug_roll_integrator,0.2);
```

```
      hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
                             velocity_vector[X]
                 + hover_aug_roll_integrator;
      hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                             hover_aug_roll_angle,
                             MAX_ATT_CTL_ANGLE);
      stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
....
  }
  else
  {
      stab_aug_roll = 0.0;
....
#ifdef notdef
      hover_aug_roll_integrator = 0.0;      /* added 8/31/89 (jwc) */
      hover_aug_pitch_integrator = 0.0;
#endif
  }
   controller_cyclic_roll = cyclic_roll + stab_aug_roll;
....
```

See APPENDIX B for a complete source code listing.

3.1.59 HOVER_AUG_ROLL_I_GAIN

HOVER_AUG_ROLL_I_GAIN is a constant defining the slope for the hover augmentation roll integrator equation.

3.1.59.1 Initialization

The constant HOVER_AUG_ROLL_I_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_ROLL_I_GAIN                    aero_data[58]
```

3.1.59.2 Usage

During real-time execution, this variable is not recomputed.

3.1.59.2.1 Algorithm

HOVER_AUG_ROLL_I_GAIN is used to compute the hover augmentation roll integrator by a call to the CSU compute_stab_augmentation_gains. The integrator is used to compute the hover augmentation roll angle.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
              {
              hover_hold_turned_on = TRUE ;
....
              /* You should already be "hovering" (airspeed < 10 knots)
                for hover hold to show little visible swaying. */

              hover_aug_roll_integrator = 0.0 ;
....
              }
....
      hover_aug_roll_integrator +=
                      HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
      hover_aug_roll_integrator =
                      limiter(-0.2,hover_aug_roll_integrator,0.2);
      hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
                              velocity_vector[X]
                      + hover_aug_roll_integrator;
      hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_roll_angle,
                              MAX_ATT_CTL_ANGLE);
      stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
....
  }
  else
  {
      stab_aug_roll = 0.0;
....
#ifdef notdef
      hover_aug_roll_integrator = 0.0;      /* added 8/31/89 (jwc) */
      hover_aug_pitch_integrator = 0.0;
#endif
  }
   controller_cyclic_roll = cyclic_roll + stab_aug_roll;
....
```

See APPENDIX B for a complete source code listing.

3.1.60 HOVER_AUG_PITCH_P_GAIN

HOVER_AUG_PITCH_P_GAIN is a constant defining the slope for the hover augmentation pitch angle equation.

3.1.60.1 Initialization

The constant HOVER_AUG_PITCH_P_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_PITCH_P_GAIN          aero_data[59]
```

3.1.60.2 Usage

During real-time execution, this variable is not recomputed.

3.1.60.2.1 Algorithm

HOVER_AUG_PITCH_P_GAIN is used to compute the hover augmentation pitch angle by a call to the CSU compute_stab_augmentation_gains.

```
  if (hover_hold_state == ON)
 {
     if ( !hover_hold_turned_on )
          {
....
          hover_aug_pitch_integrator  =
                    HOVER_AUG_PITCH_RESET_VALUE  ;
....
          }

....
     hover_aug_pitch_integrator  +=
               HOVER_AUG_PITCH_I_GAIN  *  velocity_vector[Y];
     hover_aug_pitch_integrator  =
               limiter(-0.2,hover_aug_pitch_integrator,0.2);
     hover_aug_pitch_angle  =  HOVER_AUG_PITCH_P_GAIN  *
```

```
                                   velocity_vector[Y]
                          + hover_aug_pitch_integrator;
        hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                                   hover_aug_pitch_angle,
                                   MAX_ATT_CTL_ANGLE);
        stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);

....
   }
   else
   {
....
        stab_aug_pitch = 0.0;
....
#ifdef notdef
        hover_aug_roll_integrator = 0.0;       /* added 8/31/89 (jwc) */
        hover_aug_pitch_integrator = 0.0;
#endif
   }
....
   controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
....
```

See APPENDIX B for a complete source code listing.

3.1.61 HOVER_AUG_PITCH_I_GAIN

HOVER_AUG_PITCH_I_GAIN is a constant defining the slope for the hover augmentation pitch integrator equation.

3.1.61.1 Initialization

The constant HOVER_AUG_PITCH_I_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_PITCH_I_GAIN          aero_data[60]
```

3.1.61.2 Usage

During real-time execution, this variable is not recomputed.

3.1.61.2.1 Algorithm

HOVER_AUG_PITCH_I_GAIN is used to compute the hover augmentation pitch integrator by a call to the CSU compute_stab_augmentation_gains. The integrator is used to compute the hover augmentation pitch angle.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
             {
....
             hover_aug_pitch_integrator =
                         HOVER_AUG_PITCH_RESET_VALUE ;
....
             }
....
      hover_aug_pitch_integrator +=
                 HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
      hover_aug_pitch_integrator =
                 limiter(-0.2,hover_aug_pitch_integrator,0.2);
      hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                 velocity_vector[Y]
                 + hover_aug_pitch_integrator;
      hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                                  hover_aug_pitch_angle,
                                  MAX_ATT_CTL_ANGLE);
      stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
....
  }
  else
  {
....
      stab_aug_pitch = 0.0;
....
#ifdef notdef
      hover_aug_roll_integrator = 0.0;       /* added 8/31/89 (jwc) */
      hover_aug_pitch_integrator = 0.0;
#endif
  }
....
   controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
....
```

See APPENDIX B for a complete source code listing.

3.1.62 HOVER_AUG_YAW_P_GAIN

HOVER_AUG_YAW_P_GAIN is a constant defining the slope for the hover augmentation yaw angle equation.

3.1.62.1 Initialization

The constant HOVER_AUG_YAW_P_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_YAW_P_GAIN                    aero_data[61]
```

3.1.62.2 Usage

During real-time execution, this variable is not recomputed.

3.1.62.2.1 Algorithm

HOVER_AUG_YAW_P_GAIN is used to compute the hover augmentation yaw angle by a call to the CSU compute_stab_augmentation_gains.

```
 if (hover_hold_state == ON)
 {
     if ( !hover_hold_turned_on )
            {
....
            stab_aug_yaw_integrator = 0.0 ;
....
            }
....
     stab_aug_yaw_integrator -=
                   HOVER_AUG_YAW_I_GAIN  *
                   angular_velocity_vector[Z];
     if (stab_aug_yaw_integrator >  0.5) stab_aug_yaw_integrator = 0.5;
     if (stab_aug_yaw_integrator < -0.5) stab_aug_yaw_integrator = -0.5;
     stab_aug_yaw    = - HOVER_AUG_YAW_P_GAIN *
                         angular_velocity_vector[Z] +
                         stab_aug_yaw_integrator;
....
```

```
  }
  else
  {
....
      stab_aug_yaw = 0.0;
....
  }
....
   controller_tail_rotor = pedal + stab_aug_yaw;
....
```

See APPENDIX B for a complete source code listing.

3.1.63 HOVER_AUG_YAW_I_GAIN

HOVER_AUG_YAW_I_GAIN is a constant defining the slope for the hover augmentation yaw integrator equation.

3.1.63.1 Initialization

The constant HOVER_AUG_YAW_I_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_YAW_I_GAIN                    aero_data[62]
```

3.1.63.2 Usage

During real-time execution, this variable is not recomputed.

3.1.63.2.1 Algorithm

HOVER_AUG_YAW_I_GAIN is used to compute the hover augmentation yaw integrator by a call to the CSU compute_stab_augmentation_gains. The integrator is used to compute the hover augmentation yaw angle.

```
   if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
           {
....
```

3.1.64.2 Usage

During real-time execution, this variable is not recomputed.

3.1.64.2.1 Algorithm

HOVER_AUG_CLIMB_P_GAIN is used to compute the hover augmentation climb angle by a call to the CSU compute_stab_augmentation_gains.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
        {
....
        stab_aug_climb_integrator = 0.0 ;
....
        }
....
    stab_aug_climb_integrator -=
                HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
    if (stab_aug_climb_integrator >  0.2) stab_aug_climb_integrator = 0.2;
    if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
    stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN *
                    velocity_vector[Z] + stab_aug_climb_integrator;

    stab_aug_yaw = limiter (
                    -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                     stab_aug_yaw,
                     MAX_STAB_AUG_YAW_CLIMB_CONTROL);

    stab_aug_climb = limiter (
                    -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                     stab_aug_climb,
                     MAX_STAB_AUG_YAW_CLIMB_CONTROL);
}
```

```
  else
  {
....
      stab_aug_climb = 0.0;
....
  }
....
   controller_collective = collective + stab_aug_climb;
```

See APPENDIX B for a complete source code listing.

3.1.65 HOVER_AUG_CLIMB_I_GAIN

HOVER_AUG_CLIMB_I_GAIN is a constant defining the slope for the hover augmentation climb integrator equation.

3.1.65.1 Initialization

The constant HOVER_AUG_CLIMB_I_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  HOVER_AUG_CLIMB_I_GAIN          aero_data[64]
```

3.1.65.2 Usage

During real-time execution, this variable is not recomputed.

3.1.65.2.1 Algorithm

HOVER_AUG_CLIMB_I_GAIN is used to compute the hover augmentation climb integrator by a call to the CSU compute_stab_augmentation_gains. The integrator is used to compute the hover augmentation climb angle.

```
 if (hover_hold_state == ON)
 {
     if ( !hover_hold_turned_on )
             {
....
             stab_aug_climb_integrator = 0.0 ;
....
             }
....
     stab_aug_climb_integrator -=
                     HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
     if (stab_aug_climb_integrator >  0.2) stab_aug_climb_integrator = 0.2;
     if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
     stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN *
                             velocity_vector[Z] + stab_aug_climb_integrator;

     stab_aug_yaw = limiter (
                             -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                              stab_aug_yaw,
                              MAX_STAB_AUG_YAW_CLIMB_CONTROL);

     stab_aug_climb = limiter (
                             -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                              stab_aug_climb,
                              MAX_STAB_AUG_YAW_CLIMB_CONTROL);
 }
 else
 {
....
     stab_aug_climb = 0.0;
....
 }
....
  controller_collective = collective + stab_aug_climb;
```

See APPENDIX B for a complete source code listing.

3.1.66 MAX_STAB_AUG_PITCH_ROLL_CONTROL

MAX_STAB_AUG_PITCH_ROLL_CONTROL is a constant defining the upper and lower limits of the attitude control roll command and attitude control pitch command for cross coupling effects.

3.1.66.1 Initialization

The constant MAX_STAB_AUG_PITCH_ROLL_CONTROL is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_STAB_AUG_PITCH_ROLL_CONTROL    aero_data[65]
```

3.1.66.2 Usage

During real-time execution, this variable is not recomputed.

3.1.66.2.1 Algorithm

MAX_STAB_AUG_PITCH_ROLL_CONTROL is used to compute the upper and lower limits of the attitude control roll command by a call to the CSU set_roll_attitude.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
                (roll - angle);
/**** These used to be attitude_control_pitch_integrator instead of
    attitude_control_roll_integrator.      PJM   11-1-89
attitude_control_pitch_integrator =
                limiter (-0.1, attitude_control_pitch_integrator, 0.1);
******/
attitude_control_roll_integrator =
          limiter (-0.1, attitude_control_roll_integrator, 0.1);
attitude_control_roll_command = ATT_CTL_ROLL_P_GAIN *
                (roll - angle);
attitude_control_roll_command += attitude_control_roll_integrator;
attitude_control_roll_command =  limiter (
                -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
                attitude_control_roll_command,
                MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_roll_command);
```

MAX_STAB_AUG_PITCH_ROLL_CONTROL is used to compute the upper and lower limits of the attitude control pitch command by a call to the CSU set_pitch_attitude.

```
attitude_control_pitch_integrator +=
                ATT_CTL_PITCH_I_GAIN * (pitch - angle);
attitude_control_pitch_integrator =
        limiter (-0.1, attitude_control_pitch_integrator, 0.1);
attitude_control_pitch_command = ATT_CTL_PITCH_P_GAIN *
                (pitch - angle);
attitude_control_pitch_command += attitude_control_pitch_integrator;
attitude_control_pitch_command = limiter (
                -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
                attitude_control_pitch_command,
                MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_pitch_command);
```

See APPENDIX B for a complete source code listing.

3.1.67 MAX_STAB_AUG_YAW_CLIMB_CONTROL

MAX_STAB_AUG_YAW_CLIMB_CONTROL is a constant defining the upper and lower limits of the stabilator augmentation yaw command and stabilator augmentation climb command for cross coupling effects.

3.1.67.1 Initialization

The constant MAX_STAB_AUG_YAW_CLIMB_CONTROL is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_STAB_AUG_YAW_CLIMB_CONTROL   aero_data[66]
```

3.1.67.2 Usage

During real-time execution, this variable is not recomputed.

3.1.67.2.1 Algorithm

MAX_STAB_AUG_YAW_CLIMB_CONTROL is used to compute the upper and lower limits of the stabilator augmentation yaw command and stabilator augmentation climb command by a call to the CSU compute_stab_augmentation_gains.

```
 if (hover_hold_state == ON)
 {
     if ( !hover_hold_turned_on )
             {
....
             stab_aug_climb_integrator = 0.0 ;
....
             }
....
     stab_aug_climb_integrator -=
                     HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
     if (stab_aug_climb_integrator >  0.2) stab_aug_climb_integrator = 0.2;
     if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
     stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN *
                         velocity_vector[Z] + stab_aug_climb_integrator;

     stab_aug_yaw = limiter (
                         -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                          stab_aug_yaw,
                          MAX_STAB_AUG_YAW_CLIMB_CONTROL);

     stab_aug_climb = limiter (
                         -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                          stab_aug_climb,
                          MAX_STAB_AUG_YAW_CLIMB_CONTROL);
 }
 else
 {
....
     stab_aug_climb = 0.0;
....
 }
....
  controller_collective = collective + stab_aug_climb;
```

See APPENDIX B for a complete source code listing.

3.1.68 ROLL_RATE_DAMPING_GAIN

ROLL_RATE_DAMPING_GAIN is a constant defining the roll damping rate.

3.1.68.1 Initialization

The constant ROLL_RATE_DAMPING_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define ROLL_RATE_DAMPING_GAIN          aero_data[67]
```

3.1.68.2 Usage

During real-time execution, this variable is not recomputed.

3.1.68.2.1 Algorithm

ROLL_RATE_DAMPING_GAIN is used to initialize the roll damping by a call to the CSU aerodyn_init.

```
roll_damping = ROLL_RATE_DAMPING_GAIN;
```

The roll damping is increased if hover hold is turned on by a call to the CSU compute_stab_augmentation_gains.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
              {
              hover_hold_turned_on = TRUE ;

              roll_damping = 2 * ROLL_RATE_DAMPING_GAIN;
....
              }
....
  }
```

```
  else
  {
....
      roll_damping =  ROLL_RATE_DAMPING_GAIN;
....
  }
```

The roll_damping is used to compute the y-axis element of damping of the body moment vector by a call to the CSU compute_body_damping_forces_and_moments.

```
moment_body_damping[Y] = - roll_damping * roll_rate;
```

See APPENDIX B for a complete source code listing.

3.1.69 PITCH_RATE_DAMPING_GAIN

PITCH_RATE_DAMPING_GAIN is a constant defining the pitch damping rate.

3.1.69.1 Initialization

The constant PITCH_RATE_DAMPING_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define PITCH_RATE_DAMPING_GAIN          aero_data[68]
```

3.1.69.2 Usage

During real-time execution, this variable is not recomputed.

3.1.69.2.1 Algorithm

PITCH_RATE_DAMPING_GAIN is used to initialize the pitch damping by a call to the CSU aerodyn_init.

```
pitch_damping = PITCH_RATE_DAMPING_GAIN;
```

The pitch damping is increased if hover hold is turned on by a call to the CSU compute_stab_augmentation_gains.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
        {
....
        pitch_damping = 2 * PITCH_RATE_DAMPING_GAIN;
                    /* jwc 8/90 */
....
        }
....
}
else
{
....
    pitch_damping = PITCH_RATE_DAMPING_GAIN;  /* jwc 8/90 */
....
}
```

The pitch_damping is used to compute the x-axis element of damping of the body moment vector by a call to the CSU compute_body_damping_forces_and_moments.

```
moment_body_damping[X] = - pitch_damping * pitch_rate;
```

See APPENDIX B for a complete source code listing.

3.1.70 YAW_RATE_DAMPING_GAIN

YAW_RATE_DAMPING_GAIN is a constant defining the yaw damping rate.

3.1.70.1 Initialization

The constant YAW_RATE_DAMPING_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU

aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define  YAW_RATE_DAMPING_GAIN                    aero_data[69]
```

3.1.70.2 Usage

During real-time execution, this variable is not recomputed.

3.1.70.2.1 Algorithm

YAW_RATE_DAMPING_GAIN is used to initialize the yaw damping by a call to the CSU aerodyn_init.

```
yaw_damping = YAW_RATE_DAMPING_GAIN;
```

The yaw_damping is used to compute the z-axis element of damping of the body moment vector by a call to the CSU compute_body_damping_forces_and_moments.

```
moment_body_damping[Z] = - yaw_damping * yaw_rate;
```

See APPENDIX B for a complete source code listing.

3.1.71 VERTICAL_RATE_DAMPING_GAIN

VERTICAL_RATE_DAMPING_GAIN is a constant defining the vertical damping rate.

3.1.71.1 Initialization

The constant VERTICAL_RATE_DAMPING_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define VERTICAL_RATE_DAMPING_GAIN          aero_data[70]
```

3.1.71.2 Usage

During real-time execution, this variable is not recomputed.

3.1.71.2.1 Algorithm

VERTICAL_RATE_DAMPING_GAIN is used to compute the z-axis element of damping on the body force by a call to the CSU compute_body_damping_forces_and_moments.

```
force_body_damping[Z] = -velocity_vector[Z] *
                              VERTICAL_RATE_DAMPING_GAIN;
```

See APPENDIX B for a complete source code listing.

3.1.72 LATERAL_VELOCITY_DAMPING_GAIN

LATERAL_VELOCITY_DAMPING_GAIN is a constant defining the lateral velocity damping rate.

3.1.72.1 Initialization

The constant LATERAL_VELOCITY_DAMPING_GAIN is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define LATERAL_VELOCITY_DAMPING_GAIN       aero_data[71]
```

3.1.72.2 Usage

During real-time execution, this variable is not recomputed.

3.1.72.2.1 Algorithm

LATERAL_VELOCITY_DAMPING_GAIN is used to compute the x-axis element of damping on the body force by a call to the CSU compute_body_damping_forces_and_moments.

```
force_body_damping[X] = -velocity_vector[X] *
                        LATERAL_VELOCITY_DAMPING_GAIN;
```

See APPENDIX B for a complete source code listing.

3.1.73 LIFT_COEFF_VIRTUAL_WING

LIFT_COEFF_VIRTUAL_WING is a constant defining the lift coefficient of the virtual wing.

3.1.73.1 Initialization

The constant LIFT_COEFF_VIRTUAL_WING is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define LIFT_COEFF_VIRTUAL_WING        aero_data[72]
```

3.1.73.2 Usage

During real-time execution, this variable is not recomputed.

3.1.73.2.1 Algorithm

LIFT_COEFF_VIRTUAL_WING is used to initialize the lift coefficient of the virtual wing by a call to the CSU compute_lift_drag_coefficients.

```
lift_coefficient_virtual_wing = LIFT_COEFF_VIRTUAL_WING;
```

The lift_coefficient_virtual_wing is used to compute the lift force on the virtual wing by a call to the CSU compute_lift_drag_forces.

```
        lift_coefficient_virtual_wing * VIRTUAL_WING_AREA;
```

See APPENDIX B for a complete source code listing.

3.1.74 OSWALD_EFFIC_FACTOR

OSWALD_EFFIC_FACTOR is a constant defining the oswald efficiency factor.

3.1.74.1 Initialization

The constant OSWALD_EFFIC_FACTOR is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define OSWALD_EFFIC_FACTOR                aero_data[73]
```

3.1.74.2 Usage

During real-time execution, this variable is not recomputed.

3.1.74.2.1 Algorithm

OSWALD_EFFIC_FACTOR is used to initialize the oswald efficiency factor by a call to the CSU compute_lift_drag_coefficients.

```
oswald_efficiency_factor = OSWALD_EFFIC_FACTOR;
```

See APPENDIX B for a complete source code listing.

3.1.75 INDUCED_DRAG_COEFF

INDUCED_DRAG_COEFF is a constant defining the induced drag coefficient.

3.1.75.1 Initialization

The constant INDUCED_DRAG_COEFF is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define INDUCED_DRAG_COEFF                    aero_data[74]
```

3.1.75.2 Usage

During real-time execution, this variable is not recomputed.

3.1.75.2.1 Algorithm

INDUCED_DRAG_COEFF is used to initialize the induced drag coefficient by a call to the CSU compute_lift_drag_coefficients.

```
induced_drag_coefficient = INDUCED_DRAG_COEFF;
```

The induced_drag_coefficient is used to compute the total incompressible drag by a call to the CSU compute_lift_drag_coefficients.

```
total_incompressible_drag_coefficient = parasite_drag_coefficient +
                                        induced_drag_coefficient;
```

See APPENDIX B for a complete source code listing.

3.2 Aero_init

This data array consists of initial values for positions of the control inputs, stabilator augmentation integrators, attitude control integrators, and hover augmentation integrators.

3.2.1 Cyclic_pitch

Cyclic_pitch is a variable defining the longitudinal position of the cyclic.

3.2.1.1 Initialization

The variable cyclic_pitch is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
cyclic_pitch =                    aero_init[ 0];
```

3.2.1.2 Usage

During real-time execution, this variable is recomputed. The normal range is from -1.0 to +1.0.

3.2.1.2.1 Algorithm

Cyclic_pitch is used to compute the cyclic controller pitch input. The longitudinal position of the cyclic is set by a call to the CSU aerodyn_set_longitudinal_stick to read the physical position.

```
void aerodyn_set_longitudinal_stick (val)
   REAL val;
{
   cyclic_pitch = -val;
}
```

The controller_cyclic_pitch is used to compute the pitching moment of the moment_body_main_rotor vector during execution of CSU compute_rotor_forces_and_moments.

```
moment_body_main_rotor[X] =
          - controller_cyclic_pitch *
          MAIN_ROTOR_MAX_PITCH_MOMENT;
```

The cyclic_pitch is used to compute the controller_cyclic_pitch in the CSC compute_stab_augmentation_gains.

```
controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
```

During execution of the simple flight model, the cyclic_pitch is used to compute the lift_factor in the CSC aerodyn_simple_simul.

```
lift_factor = velocity_vector[1] * velocity_vector[1] * H_CL *
            - cyclic_pitch;
```

During execution of the simple flight model, the cyclic_pitch is used to compute the pitch element of the desired orientation vector and the torque required in the CSC aerodyn_simple_simul.

```
orient_vec[0] = H_KPR * - cyclic_pitch + hover_hold_additions[0];
```

During execution of the stealth flight model, the cyclic_pitch is used to compute the desired rotation vector and the desired linear velocity vector in the CSC aerodyn_stealth_simul.

```
if (hover_hold_state == ON)
{  /* no linear velocity in X,Y, only pitch */
     desired_lin_vel[X] = desired_lin_vel[Y] = 0.0;
     desired_rot_vel[X] = -cyclic_pitch * cyclic_pitch * sign(cyclic_pitch);
     desired_rot_vel[Y] = 0.0;
}
else
{
     if (level_view)/* when not in pitch mode, level view */
     {
        vehicle_set_orientation_matrix (level); /* identity matrix */
        vehicle_set_orientation (kinematics_get_heading());
        level_view = FALSE;
     }

     desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
        * H_SIDE_MUL;
     desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
        * H_FWD_MUL;

     desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
}
```

See APPENDIX B for a complete source code listing.

3.2.2 Cyclic_roll

Cyclic_roll is a variable defining the lateral position of the cyclic.

3.2.2.1 Initialization

The variable cyclic_roll is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
cyclic_roll =                    aero_init[ 1];
```

3.2.2.2 Usage

During real-time execution, this variable is recomputed. The normal range is from -1.0 to +1.0.

3.2.2.2.1 Algorithm

Cyclic_roll is used to compute the cyclic controller roll input. The lateral position of the cyclic is set by a call to the CSU aerodyn_set_lateral_stick to read the physical position.

```
void aerodyn_set_lateral_stick (val)
   REAL val;
{
   cyclic_roll = -val;
}
```

The controller_cyclic_roll is used to compute the rolling moment of the moment_body_main_rotor vector during execution of CSU compute_rotor_forces_and_moments.

```
moment_body_main_rotor[Y] =
            controller_cyclic_roll *
            MAIN_ROTOR_MAX_ROLL_MOMENT;
```

The cyclic_roll is used to compute the controller_cyclic_roll in the CSC compute_stab_augmentation_gains.

```
controller_cyclic_roll = cyclic_roll + stab_aug_roll;
```

During execution of the simple flight model, the cyclic_roll is used to compute the roll element of the desired orientation vector and the torque required in the CSC aerodyn_simple_simul.

```
orient_vec[1] = H_KPR * cyclic_roll + hover_hold_additions[1];
```

During execution of the stealth flight model, the cyclic_roll is used to compute the desired rotation vector and the desired linear velocity vector in the CSC aerodyn_stealth_simul.

```
if (hover_hold_state == ON)
{ /* no linear velocity in X,Y, only pitch */
    desired_lin_vel[X] = desired_lin_vel[Y] = 0.0;
    desired_rot_vel[X] = -cyclic_pitch * cyclic_pitch * sign(cyclic_pitch);
    desired_rot_vel[Y] = 0.0;
}
else
{
    if (level_view)/* when not in pitch mode, level view */
    {
        vehicle_set_orientation_matrix (level); /* identity matrix */
        vehicle_set_orientation (kinematics_get_heading());
        level_view = FALSE;
    }

    desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
        * H_SIDE_MUL;
    desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
        * H_FWD_MUL;

    desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
}
```

See APPENDIX B for a complete source code listing.

3.2.3 Collective

Collective is a variable defining the position of the collective.

3.2.3.1 Initialization

The variable collective is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
if (selected_model != STEALTH_MODEL)
  collective =                    aero_init[ 2];
else
{
  collective = 0.5;
    allow_takeoff = TRUE;
}
```

3.2.3.2 Usage

During real-time execution, this variable is recomputed. The normal range is from 0.0 to +1.0.

3.2.3.2.1 Algorithm

Collective is used to compute the collective controller input. The collective position of the cyclic is set by a call to the CSU aerodyn_set_collective to read the physical position.

```
void aerodyn_set_collective (val)
  REAL val;
{
   if (funny_little_kludge)
       collective = log10 (val * 9.0 + 1.0); /* or, how to make linear log */
  else
     collective = val;
}
```

Controller_collective is used to compute rotor loads during execution of the CSU compute_rotor_loads.

```
main_rotor_load_torque = controller_collective *
                    MAIN_ROTOR_MAX_LOAD_TORQUE;
```

Controller_collective is used to compute main_rotor_thrust during execution of the CSU compute_rotor_forces_and_moments.

```
main_rotor_thrust = powertrain_percent_shaft_speed *
                    controller_collective
                    * MAIN_ROTOR_MAX_THRUST;
```

The collective is used to compute the controller_collective in the CSC compute_stab_augmentation_gains.

```
controller_collective = collective + stab_aug_climb;
```

During execution of the simple flight model, collective is used to compute a collective factor, which in turn is used to compute power in the CSC aerodyn_simple_simul.

```
coll_factor = max(0.0,collective - 0.3);
power = H_KP * coll_factor + hover_hold_additions[2];
power +=  gross_weight * collective/(H_K2+collective) * 1.25;
power = min (MAX_HELICOPTER_POWER, power);
power = max (0.0, power);
```

During execution of the stealth flight model, collective is adjusted for dead zone and for -1 to 1 range.

```
adj_collective = (collective - 0.5) * 2.0;   /* change to -1 to 1 */
```

During execution of the stealth flight model, the adjusted collective input is limited during allow_takeoff state.

```
if (allow_takeoff)
{
    if (adj_collective > 0.0)
    {
        allow_takeoff = FALSE;
    }
    else
    {
        adj_collective = 0.0;
    }
}
```

During execution of the stealth flight model, the adjusted collective input is used to compute the vertical component of the desired linear velocity vector.

```
desired_lin_vel[Z] = adj_collective * adj_collective *
    sign (adj_collective ) * H_COLL_MUL;
```

See APPENDIX B for a complete source code listing.

3.2.4 Pedal

Pedal is a variable defining the position of the pedals (yaw control).

3.2.4.1 Initialization

The variable pedal is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
pedal =                         aero_init[ 3];
```

3.2.4.2 Usage

During real-time execution, this variable is recomputed. The normal range is from -1.0 to +1.0.

3.2.4.2.1 Algorithm

Pedal is used to compute the tail rotor controller input. The position of the pedals is set by a call to the CSU aerodyn_set_pedal to read the physical position.

```
void aerodyn_set_pedal (val)
   REAL val;
{
   pedal = val;
}
```

The pedal is used to compute the controller_tail_rotor in the CSC compute_stab_augmentation_gains.

```
controller_tail_rotor = pedal + stab_aug_yaw;
```

During execution of the simple flight model, the pedal is used to compute the yaw element of the desired orientation vector in the CSC aerodyn_simple_simul.

```
orient_vec[2] = kinematics_get_yaw () + sign(pedal) * pedal
            * pedal * H_KY;
```

During execution of the stealth flight model, the pedal is used to compute vertical element of the desired rotation vector in the CSC aerodyn_stealth_simul.

```
desired_rot_vel[Z] =  pedal * pedal * sign(pedal);
```

See APPENDIX B for a complete source code listing.

3.2.5 Stab_aug_pitch_integrator

Stab_aug_pitch_integrator is a variable defining the integrator for the stabilization augmentation pitch axis.

3.2.5.1 Initialization

The variable stab_aug_pitch_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
stab_aug_pitch_integrator = aero_init[ 4];
```

3.2.5.2 Usage

During real-time execution, this variable is not used.

See APPENDIX B for a complete source code listing.

3.2.6 Stab_aug_roll_integrator

Stab_aug_roll_integrator is a variable defining the integrator for the stabilization augmentation roll axis.

3.2.6.1 Initialization

The variable stab_aug_roll_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
stab_aug_roll_integrator =          aero_init[ 5];
```

3.2.6.2 Usage

During real-time execution, this variable is not used.

See APPENDIX B for a complete source code listing.

3.2.7 Stab_aug_yaw_integrator

Stab_aug_yaw_integrator is a variable defining the integrator for the stabilization augmentation yaw axis.

3.2.7.1 Initialization

The variable stab_aug_yaw_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
stab_aug_yaw_integrator =          aero_init[ 6];
```

3.2.7.2 Usage

During real-time execution, this variable is recomputed.

3.2.7.2.1 Algorithm

The stab_aug_yaw_integrator is computed and limited, then used to compute the stab_aug_yaw and the tail rotor control input in the CSC compute_stab_augmentation_gains.

```
 if (hover_hold_state == ON)
 {
     if ( !hover_hold_turned_on )
            {
            hover_hold_turned_on = TRUE ;
....
            /* You should already be "hovering" (airspeed < 10 knots)
              for hover hold to show little visible swaying. */
....
            stab_aug_yaw_integrator = 0.0 ;
....
 ,   stab_aug_yaw_integrator -=
            HOVER_AUG_YAW_I_GAIN * angular_velocity_vector[Z];
     if (stab_aug_yaw_integrator >  0.5) stab_aug_yaw_integrator = 0.5;
     if (stab_aug_yaw_integrator < -0.5) stab_aug_yaw_integrator = -0.5;
     stab_aug_yaw    = - HOVER_AUG_YAW_P_GAIN *
            angular_velocity_vector[Z] + stab_aug_yaw_integrator;

....
     stab_aug_yaw = limiter (
            -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
            stab_aug_yaw,
            MAX_STAB_AUG_YAW_CLIMB_CONTROL);
....
 }
 else
 {
....
     stab_aug_yaw = 0.0;
....
 }
....
 controller_tail_rotor = pedal + stab_aug_yaw;
....
}
```

See APPENDIX B for a complete source code listing.

3.2.8 Stab_aug_climb_integrator

Stab_aug_climb_integrator is a variable defining the integrator for the stabilization augmentation climb axis.

3.2.8.1 Initialization

The variable stab_aug_climb_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
stab_aug_climb_integrator =          aero_init[ 7];
```

3.2.8.2 Usage

During real-time execution, this variable is recomputed.

3.2.8.2.1 Algorithm

The stab_aug_climb_integrator is computed and limited, then used to compute the stab_aug_climb and the tail rotor control input in the CSC compute_stab_augmentation_gains.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
        {
        hover_hold_turned_on = TRUE ;
....
        /* You should already be "hovering" (airspeed < 10 knots)
          for hover hold to show little visible swaying. */
....
        stab_aug_climb_integrator = 0.0 ;
....
    stab_aug_climb_integrator -=
        HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
    if (stab_aug_climb_integrator >  0.2) stab_aug_climb_integrator = 0.2;
    if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
    stab_aug_climb    = - HOVER_AUG_CLIMB_P_GAIN *
        velocity_vector[Z] + stab_aug_climb_integrator;
....
    stab_aug_climb = limiter (
```

```
            -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
            stab_aug_climb,
            MAX_STAB_AUG_YAW_CLIMB_CONTROL);
....
  }
  else
  {
....
     stab_aug_climb = 0.0;
....
  }
....
   controller_collective = collective + stab_aug_climb;
....
}
```

See APPENDIX B for a complete source code listing.

3.2.9 Attitude_control_pitch_integrator

Attitude_control_pitch_integrator is a variable defining the integrator for the attitude control pitch axis.

3.2.9.1 Initialization

The variable attitude_control_pitch_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
attitude_control_pitch_integrator = aero_init[ 8];
```

3.2.9.2 Usage

During real-time execution, this variable is recomputed.

3.2.9.2.1 Algorithm

The attitude_control_pitch_integrator is computed and limited, then used to compute the attitude_control_pitch_command in the CSC set_pitch_attitude.

```
attitude_control_pitch_integrator +=
        ATT_CTL_PITCH_I_GAIN * (pitch - angle);
attitude_control_pitch_integrator =
        limiter (-0.1, attitude_control_pitch_integrator, 0.1);
attitude_control_pitch_command = ATT_CTL_PITCH_P_GAIN *
        (pitch - angle);
attitude_control_pitch_command += attitude_control_pitch_integrator;
attitude_control_pitch_command = limiter (
        -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
        attitude_control_pitch_command,
        MAX_STAB_AUG_PITCH_ROLL_CONTROL);
```

See APPENDIX B for a complete source code listing.

3.2.10 Attitude_control_roll_integrator

Attitude_control_roll_integrator is a variable defining the integrator for the attitude control roll axis.

3.2.10.1 Initialization

The variable attitude_control_roll_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
attitude_control_roll_integrator =   aero_init[ 9];
```

3.2.10.2 Usage

During real-time execution, this variable is recomputed.

3.2.10.2.1 Algorithm

The attitude_control_roll_integrator is computed and limited, then used to compute the attitude_control_roll_command in the CSC set_roll_attitude.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
        (roll - angle);
attitude_control_roll_integrator =
        limiter (-0.1, attitude_control_roll_integrator, 0.1);
attitude_control_roll_command = ATT_CTL_ROLL_P_GAIN *
        (roll - angle);
attitude_control_roll_command += attitude_control_roll_integrator;
attitude_control_roll_command =   limiter (
        -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
        attitude_control_roll_command,
        MAX_STAB_AUG_PITCH_ROLL_CONTROL);
```

See APPENDIX B for a complete source code listing.

3.2.11 Hover_aug_pitch_integrator

Hover_aug_pitch_integrator is a variable defining the integrator for the hover augmentation pitch axis.

3.2.11.1 Initialization

The variable hover_aug_pitch_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_pitch_integrator =        aero_init[10];
```

3.2.11.2 Usage

During real-time execution, this variable is recomputed.

3.2.11.2.1 Algorithm

The hover_aug_pitch_integrator is computed and limited, then used to compute the hover_aug_pitch_angle in the CSC compute_stab_augmentation_gains.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
            {
            hover_hold_turned_on = TRUE ;
....
            /* You should already be "hovering" (airspeed < 10 knots)
              for hover hold to show little visible swaying. */
....
            hover_aug_pitch_integrator =
                        HOVER_AUG_PITCH_RESET_VALUE ;
....
      hover_aug_pitch_integrator +=
                  HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
      hover_aug_pitch_integrator =
                  limiter(-0.2,hover_aug_pitch_integrator,0.2);
      hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                  velocity_vector[Y]
                  + hover_aug_pitch_integrator;
      hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_pitch_angle,
                              MAX_ATT_CTL_ANGLE);
      stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
....
  }
  else
  {
....
#ifdef notdef
....
      hover_aug_pitch_integrator = 0.0;
#endif
....
  }
....
}
```

See APPENDIX B for a complete source code listing.

3.2.12 Hover_aug_roll_integrator

Hover_aug_roll_integrator is a variable defining the integrator for the hover augmentation roll axis.

3.2.12.1 Initialization

The variable hover_aug_roll_integrator is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_roll_integrator =         aero_init[11];
```

3.2.12.2 Usage

During real-time execution, this variable is recomputed.

3.2.12.2.1 Algorithm

The hover_aug_roll_integrator is computed and limited, then used to compute the hover_aug_roll_angle in the CSC compute_stab_augmentation_gains.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
        {
        hover_hold_turned_on = TRUE ;
....
        /* You should already be "hovering" (airspeed < 10 knots)
          for hover hold to show little visible swaying. */
....
        hover_aug_pitch_integrator = 0.0 ;
....
        hover_aug_roll_integrator +=
                HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
        hover_aug_roll_integrator =
                limiter(-0.2,hover_aug_roll_integrator,0.2);
        hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
                velocity_vector[X]
                + hover_aug_roll_integrator;
        hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                        hover_aug_roll_angle,
                        MAX_ATT_CTL_ANGLE);
        stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
....
```

```
  }
  else
  {
....
#ifdef notdef
....
      hover_aug_roll_integrator = 0.0;
#endif
....
  }
....
}
```

See APPENDIX B for a complete source code listing.

3.2.13 Hover_aug_pitch_angle

Hover_aug_pitch_angle is a variable defining the integrator for the stabilization augmentation climb axis.

3.2.13.1 Initialization

The variable hover_aug_pitch_angle is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_pitch_angle =            aero_init[12];
```

3.2.13.2 Usage

During real-time execution, this variable is recomputed.

3.2.13.2.1 Algorithm

The hover_aug_pitch_angle is computed from the hover_aug_pitch_integrator and y-element of the velocity_vector in the CSC compute_stab_augmentation_gains.

```
  if (hover_hold_state == ON)
  {
      if ( !hover_hold_turned_on )
            {
            hover_hold_turned_on = TRUE ;
....
            /* You should already be "hovering" (airspeed < 10 knots)
              for hover hold to show little visible swaying. */
....
            hover_aug_pitch_integrator =
                        HOVER_AUG_PITCH_RESET_VALUE ;
....
      hover_aug_pitch_integrator +=
                  HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
      hover_aug_pitch_integrator =
                  limiter(-0.2,hover_aug_pitch_integrator,0.2);
      hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                  velocity_vector[Y]
                  + hover_aug_pitch_integrator;
      hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_pitch_angle,
                              MAX_ATT_CTL_ANGLE);
      stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
....
  }
  else
  {
....
#ifdef notdef
....
      hover_aug_pitch_integrator = 0.0;
#endif
....
  }
....
}
```

See APPENDIX B for a complete source code listing.

3.2.14 Hover_aug_roll_angle

Hover_aug_roll_angle is a variable defining the integrator for the stabilization augmentation climb axis.

3.2.14.1 Initialization

The variable hover_aug_roll_angle is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_roll_angle =            aero_init[13];
```

3.2.14.2 Usage

During real-time execution, this variable is recomputed.

3.2.14.2.1 Algorithm

The hover_aug_roll_angle is computed from the hover_aug_roll_integrator and the x-element of the velocity_vector in the CSC compute_stab_augmentation_gains.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
        {
        hover_hold_turned_on = TRUE ;
....
        /* You should already be "hovering" (airspeed < 10 knots)
          for hover hold to show little visible swaying. */
....
        hover_aug_pitch_integrator = 0.0 ;
....
        hover_aug_roll_integrator +=
             HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
        hover_aug_roll_integrator =
             limiter(-0.2,hover_aug_roll_integrator,0.2);
        hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
             velocity_vector[X]
             + hover_aug_roll_integrator;
        hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                          hover_aug_roll_angle,
                          MAX_ATT_CTL_ANGLE);
        stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
....
```

```
  }
  else
  {
....
#ifdef notdef
....
      hover_aug_roll_integrator = 0.0;
#endif
....
  }
....
}
```

See APPENDIX B for a complete source code listing.

3.3 Aero_simple

This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "simple" mode. The following constants are for the simplified dynamics model. The model is a modification of the aerodynamics model from the SAF. Global variables defined for the real aerodynamics are re-used here to allow overlap in generic routines for operations such as control inputs, init, etc.

3.3.1 MAX_HELICOPTER_POWER

MAX_HELICOPTER_POWER is a constant defining the maximum helicopter power.

3.3.1.1 Initialization

The constant MAX_HELICOPTER_POWER is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define MAX_HELICOPTER_POWER        aero_simple[ 0]
```

3.3.1.2 Usage

During real-time execution, this constant is not recomputed.

3.3.1.2.1 Algorithm

MAX_HELICOPTER_POWER is used to limit the maximum power of the vehicle during execution of the CSU aerodyn_simple_simul.

```
power = min (MAX_HELICOPTER_POWER, power);
```

See APPENDIX B for a complete source code listing.

3.3.2 MAX_HH

MAX_HH is a constant defining the maximum hover hold input.

3.3.2.1 Initialization

The constant MAX_HH is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define MAX_HH                    aero_simple[ 1]
```

3.3.2.2 Usage

During real-time execution, this constant is not recomputed.

3.3.2.2.1 Algorithm

MAX_HH is used to limit the hover hold control inputs of the vehicle during execution of the CSU aerodyn_simple_simul.

```
if (hover_hold_state == ON)
{
    hover_hold_additions[0] = min(velocity_vector[1] *
                                  H_KH,MAX_HH);
    hover_hold_additions[0] = max(hover_hold_additions[0],-MAX_HH);
    hover_hold_additions[1] = min(- velocity_vector[0] *
                                  H_KH,MAX_HH);
    hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
    hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
}
```

```
else
{
    hover_hold_additions[0] = 0;
    hover_hold_additions[1] = 0;
    hover_hold_additions[2] = 0;
}
```

See APPENDIX B for a complete source code listing.

3.3.3 H_K1

H_K1 is a constant defining the gain on position error.

3.3.3.1 Initialization

The constant H_K1 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_K1                    aero_simple[ 2]
```

3.3.3.2 Usage

During real-time execution, this constant is not recomputed.

3.3.3.2.1 Algorithm

H_K1 is used to compute the angular velocity necessary to achieve the desired orientation in exactly one tick. (delta theta/ delta T). Then get the angular acceleration needed to get to that velocity in one frame for the vehicle during execution of the CSU aerodyn_simple_simul.

```
for (i = X; i <= Z; ++i)
{
    vec_ptr[i] = ((des_ptr[i] - cur_ptr[i]) / DELTA_T / H_K1);
    angular_accel[i] = (vec_ptr[i] - angular_velocity_vector[i])
                / DELTA_T;
    res_ptr[i] = MOMENT_OF_INERTIA_X * angular_accel[i];
}
```

See APPENDIX B for a complete source code listing.

3.3.4 H_K2

H_K2 is a constant defining the gain on gravity term of power setting.

3.3.4.1 Initialization

The constant H_K2 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_K2                    aero_simple[ 3]
```

3.3.4.2 Usage

During real-time execution, this constant is not recomputed.

3.3.4.2.1 Algorithm

H_K2 is used to compute power of the stealth vehicle during execution of the CSU aerodyn_simple_simul.

```
power += gross_weight * collective/(H_K2+collective) * 1.25;
```

See APPENDIX B for a complete source code listing.

3.3.5 H_K7

H_K7 is a constant defining the air drag coefficient.

3.3.5.1 Initialization

The constant H_K7 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_K7                    aero_simple[ 4]
```

3.3.5.2 Usage

During real-time execution, this constant is not recomputed.

3.3.5.2.1 Algorithm

H_K7 is used to compute the drag force in the y-axis of the stealth vehicle during execution of the CSU aerodyn_simple_simul.

```
drag_ptr[Y] = square(cur_ptr[Y]) * H_K7;
```

See APPENDIX B for a complete source code listing.

3.3.6 H_K8

H_K8 is a constant defining the air drag coefficient.

3.3.6.1 Initialization

The constant H_K8 is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_K8                    aero_simple[ 5]
```

3.3.6.2 Usage

During real-time execution, this constant is not recomputed.

3.3.6.2.1 Algorithm

H_K8 is used to drag force in the x- and z-axes of the vehicle during execution of the CSU aerodyn_simple_simul.

```
drag_ptr[X] = square(cur_ptr[X]) * H_K8;
drag_ptr[Z] = square(cur_ptr[Z]) * H_K8;
```

See APPENDIX B for a complete source code listing.

3.3.7 H_KP

H_KP is a constant gain defining the power relationship with the collective input.

3.3.7.1 Initialization

The constant H_KP is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_KP                aero_simple[ 6]
```

3.3.7.2 Usage

During real-time execution, this constant is not recomputed.

3.3.7.2.1 Algorithm

H_KP is used to compute power of the stealth vehicle during execution of the CSU aerodyn_simple_simul.

```
power = H_KP * coll_factor + hover_hold_additions[2];
```

See APPENDIX B for a complete source code listing.

3.3.8 H_KPR

H_KPR is a constant defining the pitch/roll constant, approximately pi/3.

3.3.8.1 Initialization

The constant H_KPR is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done

only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_KPR                    aero_simple[ 7]
```

3.3.8.2 Usage

During real-time execution, this constant is not recomputed.

3.3.8.2.1 Algorithm

H_KPR is used to compute the torque required to achieve the desired orientation in pitch and roll of the vehicle during execution of the CSU aerodyn_simple_simul.

```
orient_vec[0] = H_KPR * - cyclic_pitch + hover_hold_additions[0];
orient_vec[1] = H_KPR * cyclic_roll + hover_hold_additions[1];
```

See APPENDIX B for a complete source code listing.

3.3.9 H_KY

H_KY is a constant defining the yaw constant, approximately pi/2.

3.3.9.1 Initialization

The constant H_KY is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_KY                     aero_simple[ 8]
```

3.3.9.2 Usage

During real-time execution, this constant is not recomputed.

3.3.9.2.1 Algorithm

H_KY is used to compute the torque required to achieve the desired orientation in yaw of the vehicle during execution of the CSU aerodyn_simple_simul.

```
orient_vec[2] = kinematics_get_yaw () + sign(pedal) * pedal
          * pedal * H_KY;
```

See APPENDIX B for a complete source code listing.

3.3.10 H_KH

H_KH is a constant defining the hover hold gain on velocity term.

3.3.10.1 Initialization

The constant H_KH is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_KH                    aero_simple[ 9]
```

3.3.10.2 Usage

During real-time execution, this constant is not recomputed.

3.3.10.2.1 Algorithm

H_KH is used to modify the computed limit on the velocity vector as an input to hover hold of the vehicle during execution of the CSU aerodyn_simple_simul.

```
if (hover_hold_state == ON)
{
    hover_hold_additions[0] = min(velocity_vector[1] *
                                    H_KH,MAX_HH);
    hover_hold_additions[0] = max(hover_hold_additions[0],-MAX_HH);
    hover_hold_additions[1] = min(- velocity_vector[0] *
```

```
                                    H_KH,MAX_HH);
     hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
     hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
}
else
{
     hover_hold_additions[0] = 0;
     hover_hold_additions[1] = 0;
     hover_hold_additions[2] = 0;
}
```

See APPENDIX B for a complete source code listing.

3.3.11 H_CHH

H_CHH is a constant defining the collective hover hold gain.

3.3.11.1 Initialization

The constant H_CHH is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_CHH                    aero_simple[10]
```

3.3.11.2 Usage

During real-time execution, this constant is not recomputed.

3.3.11.2.1 Algorithm

H_CHH is used to computed of the velocity vector as an input to hover hold of the vehicle during execution of the CSU aerodyn_simple_simul.

```
 if (hover_hold_state == ON)
{
     hover_hold_additions[0] = min(velocity_vector[1] *
                                    H_KH,MAX_HH);
     hover_hold_additions[0] = max(hover_hold_additions[0],-MAX_HH);
     hover_hold_additions[1] = min(- velocity_vector[0] *
                                    H_KH,MAX_HH);
```

```
        hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
        hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
    }
    else
    {
        hover_hold_additions[0] = 0;
        hover_hold_additions[1] = 0;
        hover_hold_additions[2] = 0;
    }
```

See APPENDIX B for a complete source code listing.

3.3.12 H_CL

H_CL is a constant defining the coefficient of lift.

3.3.12.1 Initialization

The constant H_CL is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_CL                    aero_simple[11]
```

3.3.12.2 Usage

During real-time execution, this constant is not recomputed.

3.3.12.2.1 Algorithm

H_CL is used to compute the lift as a function of cyclic pitch and velocity of the vehicle during execution of the CSU aerodyn_simple_simul.

```
lift_factor = velocity_vector[1] * velocity_vector[1] * H_CL *
              - cyclic_pitch;
```

See APPENDIX B for a complete source code listing.

3.4 Aero_stealth

This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "stealth" mode. The following is for the simplified model incorporating the stealth dynamics. In this model, the cyclic changes the desired velocity.

3.4.1 H_FWD_MUL

H_FWD_MUL is a constant defining the slope of the cyclic pitch position squared versus forward velocity curve.

3.4.1.1 Initialization

The constant H_FWD_MUL is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define H_FWD_MUL   aero_stealth[ 0]
```

3.4.1.2 Usage

During real-time execution, this constant is not recomputed.

3.4.1.2.1 Algorithm

H_FWD_MUL is used to compute the desired linear velocity in the forward direction for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
    * H_FWD_MUL;
```

See APPENDIX B for a complete source code listing.

3.4.2 H_SIDE_MUL

H_SIDE_MUL is a constant defining the slope of the cyclic roll position squared versus sideward velocity curve.

3.4.2.1 Initialization

The constant H_SIDE_MUL is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define H_SIDE_MUL    aero_stealth[ 1]
```

3.4.2.2 Usage

During real-time execution, this constant is not recomputed.

3.4.2.2.1 Algorithm

H_SIDE_MUL is used to compute the desired linear velocity in the sideward direction for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
    desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
       * H_SIDE_MUL;
```

See APPENDIX B for a complete source code listing.

3.4.3 H_COLL_MUL

H_COLL_MUL is a constant defining the slope of the collective position squared versus vertical velocity curve.

3.4.3.1 Initialization

The constant H_COLL_MUL is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define H_COLL_MUL   aero_stealth[ 2]
```

3.4.3.2 Usage

During real-time execution, this constant is not recomputed.

3.4.3.2.1 Algorithm

H_COLL_MUL is used to compute the desired linear velocity in the vertical direction for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
desired_lin_vel[Z] = adj_collective * adj_collective *
    sign (adj_collective ) * H_COLL_MUL;
```

See APPENDIX B for a complete source code listing.

3.4.4 MAX_TORQUE

MAX_TORQUE is a constant defining the maximum controller torque for the stealth vehicle.

3.4.4.1 Initialization

The constant MAX_TORQUE is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define MAX_TORQUE  aero_stealth[ 3]
```

3.4.4.2 Usage

During real-time execution, this constant is not recomputed.

3.4.4.2.1 Algorithm

MAX_TORQUE is used to limit the controller torque for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
moment_body[X] = min (MAX_TORQUE, moment_body[X]);
moment_body[Y] = min (MAX_TORQUE, moment_body[Y]);
moment_body[Z] = min (MAX_TORQUE, moment_body[Z]);

moment_body[X] = max (-MAX_TORQUE, moment_body[X]);
moment_body[Y] = max (-MAX_TORQUE, moment_body[Y]);
moment_body[Z] = max (-MAX_TORQUE, moment_body[Z]);
```

See APPENDIX B for a complete source code listing.

3.4.5 MAX_FORCE

MAX_FORCE is a constant defining the maximum controller force for the stealth vehicle.

3.4.5.1 Initialization

The constant MAX_FORCE is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define MAX_FORCE     aero_stealth[ 4]
```

3.4.5.2 Usage

During real-time execution, this constant is not recomputed.

3.4.5.2.1 Algorithm

MAX_FORCE is used to limit the controller forces for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
force_body[X] = min (MAX_FORCE, force_body[X]);
force_body[Y] = min (MAX_FORCE, force_body[Y]);
force_body[Z] = min (MAX_FORCE, force_body[Z]);

force_body[X] = max (-MAX_FORCE, force_body[X]);
force_body[Y] = max (-MAX_FORCE, force_body[Y]);
force_body[Z] = max (-MAX_FORCE, force_body[Z]);
```

See APPENDIX B for a complete source code listing.

3.4.6 MASS

MASS is a constant defining the

3.4.6.1 Initialization

The constant MASS is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define MASS            aero_stealth[ 5]
```

3.4.6.2 Usage

During real-time execution, this constant is not recomputed.

3.4.6.2.1 Algorithm

MASS is used to compute the controller forces for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
force_body[X] = (desired_lin_vel[X] - velocity_vector[X])
        * MASS/DELTA_T;
force_body[Y] = (desired_lin_vel[Y] - velocity_vector[Y])
        * MASS/DELTA_T;
force_body[Z] = (desired_lin_vel[Z] - velocity_vector[Z])
        * MASS/DELTA_T;
```

See APPENDIX B for a complete source code listing.

3.4.7 INERTIA

INERTIA is a constant defining the inertia of the stealth vehicle.

3.4.7.1 Initialization

The constant INERTIA is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define INERTIA        aero_stealth[ 6]
```

3.4.7.2 Usage

During real-time execution, this constant is not recomputed.

3.4.7.2.1 Algorithm

INERTIA is used to compute the controller torque for the stealth vehicle during execution of the CSU aerodyn_stealth_simul.

```
moment_body[X] = (desired_rot_vel[X] - angular_velocity_vector[X])
        * INERTIA/DELTA_T;
moment_body[Y] = (desired_rot_vel[Y] - angular_velocity_vector[Y])
        * INERTIA/DELTA_T;
moment_body[Z] = (desired_rot_vel[Z] - angular_velocity_vector[Z])
        * INERTIA/DELTA_T;
```

See APPENDIX B for a complete source code listing.

3.4.8 DEAD_ZONE

DEAD_ZONE is a constant defining the dead zone of the controls.

3.4.8.1 Initialization

The constant DEAD_ZONE is initialized during execution of the CSU aerodyn_init, called by CSC rwa_init. Execution of the CSU aerodyn_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define DEAD_ZONE     aero_stealth[ 7]
```

3.4.8.2 Usage

During real-time execution, this constant is not recomputed nor used.

See APPENDIX B for a complete source code listing.

3.5 Engine_data

This data array consists of characteristics and parameters describing the engine performance and control.

3.5.1 GOVERNOR_ENGINE_SPEED_SETTING

The GOVERNOR_ENGINE_SPEED_SETTING is a constant defining the maximum engine speed setting at 100 percent rpm.

3.5.1.1 Initialization

The constant GOVERNOR_ENGINE_SPEED_SETTING is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define GOVERNOR_ENGINE_SPEED_SETTING engine_data[ 0]
```

3.5.1.2 Usage

During real-time execution, this constant is not recomputed.

3.5.1.2.1 Algorithm

Engine power is computed as a function of GOVERNOR_ENGINE_SPEED_SETTING.

```
engine_power = gov_p_gain *
    (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
```

If the engine is working, GOVERNOR_ENGINE_SPEED_SETTING is used to compute the integrator_gain.

```
if (engine_status == WORKING)
{
    integrator_gain += gov_i_gain *
        (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
    if (integrator_gain > 0.5)
        integrator_gain = 0.5;
    else if (integrator_gain < -0.5)
        integrator_gain = -0.5;

    engine_power += integrator_gain;
}
else                        /* Damaged */
{
    integrator_gain = 0.0;
    if (engine_power > 0.7)
        engine_power = 0.7;
}
```

The constant GOVERNOR_ENGINE_SPEED_SETTING is used to compute powertrain_percent_shaft_speed.

```
powertrain_percent_shaft_speed = engine_speed /
    GOVERNOR_ENGINE_SPEED_SETTING;
```

See APPENDIX C for a complete source code listing.

3.5.2 GOVERNOR_P_GAIN

The GOVERNOR_P_GAIN is a constant defining the maximum engine speed gain.

3.5.2.1 Initialization

The constant GOVERNOR_P_GAIN is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define GOVERNOR_P_GAIN                    engine_data[ 1]
```

3.5.2.2 Usage

During real-time execution, this constant is not recomputed.

3.5.2.2.1 Algorithm

Engine variable gov_p_gain is initialized during execution of CSC engine_init to GOVERNOR_P_GAIN.

```
gov_p_gain =          GOVERNOR_P_GAIN;
```

The variable gov_p_gain is used to compute engine_power. The variable gov_p_gain is not recomputed during execution of CSU engine_simul.

```
engine_power = gov_p_gain *
    (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
```

See APPENDIX C for a complete source code listing.

3.5.3 GOVERNOR_I_GAIN

The GOVERNOR_I_GAIN is a constant defining the maximum engine speed gain rate of the integrator.

3.5.3.1 Initialization

The constant GOVERNOR_I_GAIN is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define GOVERNOR_I_GAIN                engine_data[ 2]
```

3.5.3.2 Usage

During real-time execution, this constant is not recomputed.

3.5.3.2.1 Algorithm

Engine variable gov_i_gain is initialized during execution of CSC engine_init to GOVERNOR_I_GAIN.

```
gov_i_gain =            GOVERNOR_I_GAIN;
```

If the engine_status is WORKING, the variable gov_i_gain is used to compute integrator_gain. The variable gov_p_gain is not recomputed during execution of CSU engine_simul.

```
if (engine_status == WORKING)
{
    integrator_gain += gov_i_gain *
        (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
    if (integrator_gain > 0.5)
        integrator_gain = 0.5;
    else if (integrator_gain < -0.5)
        integrator_gain = -0.5;

    engine_power += integrator_gain;
}
else                        /* Damaged */
{
    integrator_gain = 0.0;
    if (engine_power > 0.7)
        engine_power = 0.7;
}
```

See APPENDIX C for a complete source code listing.

3.5.4 MAX_ENGINE_TORQUE

The MAX_ENGINE_TORQUE is a constant defining the maximum engine torque.

3.5.4.1 Initialization

The constant MAX_ENGINE_TORQUE is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define MAX_ENGINE_TORQUE           engine_data[ 3]
```

3.5.4.2 Usage

During real-time execution, this constant is not recomputed.

3.5.4.2.1 Algorithm

The constant MAX_ENGINE_TORQUE is used to compute the engine_percent_torque.

```
engine_percent_torque = engine_drive_torque /
    (MAX_ENGINE_TORQUE * number_of_engines);
```

See APPENDIX C for a complete source code listing.

3.5.5 MIN_ENGINE_LOAD_TORQUE

The MIN_ENGINE_LOAD_TORQUE is a constant defining the minimum engine load torque.

3.5.5.1 Initialization

The constant MIN_ENGINE_LOAD_TORQUE is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define MIN_ENGINE_LOAD_TORQUE           engine_data[ 4]
```

3.5.5.2 Usage

During real-time execution, this constant is not recomputed.

3.5.5.2.1 Algorithm

The constant MIN_ENGINE_LOAD_TORQUE is used to set the lower limit of engine_load_torque.

```
if (engine_load_torque < MIN_ENGINE_LOAD_TORQUE)
    engine_load_torque = MIN_ENGINE_LOAD_TORQUE;
```

See APPENDIX C for a complete source code listing.

3.5.6 MAX_ENGINE_PERCENT_POWER

The MAX_ENGINE_PERCENT_POWER is a constant defining the maximum engine percent of power available.

3.5.6.1 Initialization

The constant MAX_ENGINE_PERCENT_POWER is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define MAX_ENGINE_PERCENT_POWER        engine_data[ 5]
```

3.5.6.2 Usage

During real-time execution, this constant is not recomputed.

3.5.6.2.1 Algorithm

If engine_power is greater than MAX_ENGINE_PERCENT_POWER, engine_power is limited to the MAX_ENGINE_PERCENT_POWER.

```
if (engine_power > MAX_ENGINE_PERCENT_POWER)
    engine_power = MAX_ENGINE_PERCENT_POWER;
```

See APPENDIX C for a complete source code listing.

3.5.7 ENGINE_TORQUE_INTERCEPT

The ENGINE_TORQUE_INTERCEPT is a constant defining the engine torque curve intercept for the linear engine torque equation.

3.5.7.1 Initialization

The constant ENGINE_TORQUE_INTERCEPT is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define ENGINE_TORQUE_INTERCEPT          engine_data[ 6]
```

3.5.7.2 Usage

During real-time execution, this constant is not recomputed.

3.5.7.2.1 Algorithm

The constant ENGINE_TORQUE_INTERCEPT is used to compute engine_drive_torque.

```
    engine_drive_torque = engine_power * number_of_engines *
        (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
engine_speed);
```

See APPENDIX C for a complete source code listing.

3.5.8 ENGINE_TORQUE_SLOPE

The ENGINE_TORQUE_SLOPE is a constant defining the engine torque curve slope for the linear engine torque equation.

3.5.8.1 Initialization

The constant ENGINE_TORQUE_SLOPE is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define ENGINE_TORQUE_SLOPE          engine_data[ 7]
```

3.5.8.2 Usage

During real-time execution, this constant is not recomputed.

3.5.8.2.1 Algorithm

The constant ENGINE_TORQUE_SLOPE is used to compute engine_drive_torque.

```
   engine_drive_torque = engine_power * number_of_engines *
      (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
engine_speed);
```

See APPENDIX C for a complete source code listing.

3.5.9 NOSE_GEARBOX_RATIO

The NOSE_GEARBOX_RATIO is a constant defining the nose gearbox ratio.

3.5.9.1 Initialization

The constant NOSE_GEARBOX_RATIO is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define NOSE_GEARBOX_RATIO          engine_data[ 8]
```

3.5.9.2 Usage

During real-time execution, this constant is not recomputed.

3.5.9.2.1 Algorithm

NOSE_GEARBOX_RATIO is used to compute turbine speed.

```
turbine_speed = engine_speed * NOSE_GEARBOX_RATIO;
```

See APPENDIX C for a complete source code listing.

3.5.10 MAIN_ROTOR_GEAR_RATIO

The MAIN_ROTOR_GEAR_RATIO is a constant defining the main rotor gear ratio.

3.5.10.1 Initialization

The constant MAIN_ROTOR_GEAR_RATIO is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define MAIN_ROTOR_GEAR_RATIO          engine_data[ 9]
```

3.5.10.2 Usage

During real-time execution, this constant is not recomputed.

3.5.10.2.1 Algorithm

MAIN_ROTOR_GEAR_RATIO is used to compute main_rotor_engine_load.

```
main_rotor_engine_load = main_rotor_load /
        MAIN_ROTOR_GEAR_RATIO;
```

MAIN_ROTOR_GEAR_RATIO is used to compute main_rotor_shaft_speed.

```
main_rotor_shaft_speed = engine_speed /
                MAIN_ROTOR_GEAR_RATIO;
```

MAIN_ROTOR_GEAR_RATIO is used to compute main_rotor_drive_torque.

```
main_rotor_drive_torque = (engine_drive_torque -
        tail_rotor_engine_load)
        * MAIN_ROTOR_GEAR_RATIO;
```

See APPENDIX C for a complete source code listing.

3.5.11 TAIL_ROTOR_GEAR_RATIO

The TAIL_ROTOR_GEAR_RATIO is a constant defining the tail rotor gear ratio.

3.5.11.1 Initialization

The constant TAIL_ROTOR_GEAR_RATIO is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define TAIL_ROTOR_GEAR_RATIO          engine_data[10]
```

3.5.11.2 Usage

During real-time execution, this constant is not recomputed.

3.5.11.2.1 Algorithm

TAIL_ROTOR_GEAR_RATIO is used to compute tail_rotor_engine_load.

```
tail_rotor_engine_load = tail_rotor_load / TAIL_ROTOR_GEAR_RATIO;
```

TAIL_ROTOR_GEAR_RATIO is used to compute tail_rotor_shaft_speed.

```
tail_rotor_shaft_speed = engine_speed / TAIL_ROTOR_GEAR_RATIO;
```

See APPENDIX C for a complete source code listing.

3.5.12 POWERTRAIN_INERTIA

The POWERTRAIN_INERTIA is a constant defining the powertrain inertia.

3.5.12.1 Initialization

The constant POWERTRAIN_INERTIA is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define  POWERTRAIN_INERTIA          engine_data[11]
```

3.5.12.2 Usage

During real-time execution, this constant is not recomputed.

3.5.12.2.1 Algorithm

If engine_status is WORKING, POWERTRAIN_INERTIA is used to compute engine_speed.

```
if (engine_status == WORKING)
    engine_speed += (engine_drive_torque - engine_load_torque)
        / POWERTRAIN_INERTIA;
```

See APPENDIX C for a complete source code listing.

3.5.13 MAX_FUELFLOW

The MAX_FUELFLOW is a constant defining the maximum engine fuel flow.

3.5.13.1 Initialization

The constant MAX_FUELFLOW is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

```
#define MAX_FUELFLOW                engine_data[12]
```

3.5.13.2 Usage

During real-time execution, this constant is not recomputed.

3.5.13.2.1 Algorithm

MAX_FUELFLOW is used to compute engine fuel_flow.

```
fuel_flow = engine_percent_torque * MAX_FUELFLOW;
```

See APPENDIX C for a complete source code listing.

3.6 Engine_init_data

This data array consists of initial values of the current engine state, performance, and control.

3.6.1 Engine_power

The variable engine_power is a computed variable defining the current state of engine power.

3.6.1.1 Initialization

The variable engine_power is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
engine_power =          engine_init_data[ 0]
```

3.6.1.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

If the engine is out of fuel, the engine_power is set to 0.0.

```
if (fuel_level_empty ())        /* Out of gas */
{
    engine_power = 0.0;
    engine_speed = 0.0;
}
```

The engine_power is then used to compute the engine_drive_torque.

```
engine_drive_torque = engine_power * number_of_engines *
    (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
    engine_speed);
```

See APPENDIX C for a complete source code listing.

3.6.2 Engine_percent_torque

The variable engine_percent_torque is a computed variable defining the current state of percent of engine torque.

3.6.2.1 Initialization

The variable engine_percent_torque is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
engine_percent_torque =    engine_init_data[ 1]
```

3.6.2.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.6.2.2.1 Algorithm

Percent of engine torque is computed as a function of engine_drive_torque and number of engines.

```
engine_percent_torque = engine_drive_torque /
   (MAX_ENGINE_TORQUE * number_of_engines);
```

The engine_percent_torque is used to compute the fuel_flow.

```
fuel_flow = engine_percent_torque * MAX_FUELFLOW;
```

If the engine is starting, and the engine_percent_torque is less than .0101, the engine is not starting. limited to a minimum values. If the engine is starting, and the engine_percent_torque is greater than or equal to .0101, the engine_percent_torque is limited to a value of .01.

```
if (starting_engine)
{
    if (engine_percent_torque - .01 < .0001)        /* within a delta */
       starting_engine = FALSE;
    else
       engine_percent_torque = .01;
}
```

The engine_percent_torque is output to the torque meter display.

```
meter_torque_set (engine_percent_torque);
```

The engine_percent_torque is also used to compute engine and rotor sound.

See Appendix C for a complete source code listing.

3.6.3 Engine_speed

The variable engine_speed is a computed variable defining the current state of engine speed.

3.6.3.1 Initialization

The variable engine_speed is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is

normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
engine_speed =          engine_init_data[ 2];
```

3.6.3.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.6.3.2.1 Algorithm

If the engine is working, engine_speed is computed as an incremental change each frame as a function of the difference between engine_drive_torque and engine_load_torque.

```
if (engine_status == WORKING)
    engine_speed += (engine_drive_torque - engine_load_torque)
        / POWERTRAIN_INERTIA;
```

If the engine is out of fuel, engine_speed is set to 0.0.

```
if (fuel_level_empty ())        /* Out of gas */
{
    engine_power = 0.0;
    engine_speed = 0.0;
}
```

If the engine-speed is less than 0.0, the engine_speed is limited to 0.0.

```
if (engine_speed < 0.0)
    engine_speed = 0.0;
```

If an engine is broken, engine_speed is set to 0.0.

```
void    engine_break_engine ()
{
   engine_status = BROKEN;
   engine_speed = 0.0;
   number_of_engines = 1;
}
```

If the engine is working, engine_speed is used to compute the integrator_gain for the engine_power computation.

```
if (engine_status == WORKING)
{
     integrator_gain += gov_i_gain *
        (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
     if (integrator_gain > 0.5)
        integrator_gain = 0.5;
     else if (integrator_gain < -0.5)
        integrator_gain = -0.5;

     engine_power += integrator_gain;
}
else                          /* Damaged */
{
     integrator_gain = 0.0;
     if (engine_power > 0.7)
        engine_power = 0.7;
}
```

The engine_speed is used to compute the engine_power.

```
engine_power = gov_p_gain *
   (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
```

The engine_speed is used to compute the engine_drive_torque.

```
engine_drive_torque = engine_power * number_of_engines *
    (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
engine_speed);
```

The engine_speed is used to compute the turbine_speed.

```
turbine_speed = engine_speed * NOSE_GEARBOX_RATIO;
```

The engine_speed is used to compute the main_rotor_shaft_speed.

```
main_rotor_shaft_speed = engine_speed /
MAIN_ROTOR_GEAR_RATIO;
```

The engine_speed is used to compute the tail_rotor_shaft_speed.

```
tail_rotor_shaft_speed = engine_speed / TAIL_ROTOR_GEAR_RATIO;
```

The engine_speed is used to compute the powertrain_percent_shaft_speed.

```
powertrain_percent_shaft_speed = engine_speed /
    GOVERNOR_ENGINE_SPEED_SETTING;
```

See Appendix C for a complete source code listing.

3.6.4 Integrator_gain

The variable integrator_gain is a computed variable defining the rate of change for engine_power during each frame.

3.6.4.1 Initialization

The variable integrator_gain is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. -, Engine Initialization Data Array for a summary of the initialization variable.

```
integrator_gain =        engine_init_data[ 3];
```

3.6.4.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul, if the engine is working..

3.6.4.2.1 Algorithm

The integrator_gain is computed as a function of engine_speed. It is limited to a value between 0.5 and -0.5. If the engine is not working, the integrator_gain is set to zero.

```
if (engine_status == WORKING)
{
    integrator_gain += gov_i_gain *
        (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
    if (integrator_gain > 0.5)
        integrator_gain = 0.5;
    else if (integrator_gain < -0.5)
        integrator_gain = -0.5;

    engine_power += integrator_gain;
}
else                          /* Damaged */
{
    integrator_gain = 0.0;
    if (engine_power > 0.7)
        engine_power = 0.7;
}
```

See Appendix C for a complete source code listing.

3.6.5 Last_percent_shaft_speed

The variable last_percent_shaft_speed is a computed variable defining the state of percent of powertrain shaft speed from the last frame.

3.6.5.1 Initialization

The variable last_percent_shaft_speed is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init

is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
last_percent_shaft_speed = engine_init_data[ 4];
```

3.6.5.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.6.5.2.1 Algorithm

Last_percent_shaft_speed is computed by assignment of powertrain_percent_shaft_speed after the computation of .powertrain_percent_shaft_speed during the current frame. The change of powertrain shaft speed is compared to a delta. If the absolute change of the powertrain shaft speed is greater than the delta, the sound for the rotor is recomputed.

```
if (abs (powertrain_percent_shaft_speed
        - last_percent_shaft_speed) > 0.025)
{
    /* rotor sounds depend on RPMs
     * (powertrain_percent_shaft_speed) */
    temp_percent = max (0.01, powertrain_percent_shaft_speed);
    sound_make_cont_sound  (SOUND_OF_START_ROTOR,
                SOUND_OF_VARY_ROTOR,
                SOUND_OF_STOP_ROTOR, temp_percent);
    last_percent_shaft_speed = powertrain_percent_shaft_speed;
}
```

See Appendix C for a complete source code listing.

3.6.6 Last_percent_torque

The variable last_percent_torque is a computed variable defining the state of percent of engine torque from the previous frame.

3.6.6.1 Initialization

The variable last_percent_torque is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is

normally done only once during CSCI initialization and is performed sequentially. See TABLE 3.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
last_percent_torque =     engine_init_data[ 5];
```

3.6.6.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul. It is used to compute sound change of the engine.

3.6.6.2.1 Algorithm

Last_percent_torque is computed by assignment of engine_percent_torque after the computation of .engine_percent_torque during the current frame. The change of engine torque is compared to a delta. If the absolute change of the engine torque is greater than this delta, the sound for engine is recomputed.

```
if (abs (engine_percent_torque - last_percent_torque) > 0.025)
{
    /* engine sounds depend on torque (engine_percent_torque) */
    temp_percent = max (0.01, engine_percent_torque);
    sound_make_cont_sound  (SOUND_OF_START_ENGINE,
                SOUND_OF_VARY_ENGINE,
                SOUND_OF_STOP_ENGINE, temp_percent);
    last_percent_torque = engine_percent_torque;
}
```

See Appendix C for a complete source code listing.

3.6.7 Hours_of_flight

The variable hours_of_flight is a computed variable defining the current hours of flight.

3.6.7.1 Initialization

The variable hours_of_flight is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
hours_of_flight =        engine_init_data[ 6];
```

3.6.7.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.6.7.2.1 Algorithm

Hours_of_flight is computed by incrementing the current hours_of_flight by the amount of time each frame of execution.

```
hours_of_flight += HOURS_PER_TICK;
```

See Appendix C for a complete source code listing.

3.7 Engine_stat_data

This data array consists of the initial values for flight time, engine status, number of engines, and powertrain damage status.

3.7.1 Minutes_of_flight

The variable minutes_of_flight is a computed variable defining the current minutes of flight.

3.7.1.1 Initialization

The variable minutes_of_flight is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
minutes_of_flight =       engine_stat_data[ 0];
```

3.7.1.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.1.2.1 Algorithm

Minutes_of_flight is computed as a function of the hours_of_flight.

```
minutes_of_flight = (int) (hours_of_flight * 60);
```

If a failure has occurred to the engine subsystem, the minutes_of_flight is stored in the variable old_minutes_of_flight for use in the next frame.

```
old_minutes_of_flight = minutes_of_flight;
```

See Appendix C for a complete source code listing.

3.7.2 Old_minutes_of_flight

The variable minutes_of_flight is a computed variable defining the current minutes of flight.

3.7.2.1 Initialization

The variable minutes_of_flight is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
old_minutes_of_flight =    engine_stat_data[ 1];
```

3.7.2.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.2.2.1 Algorithm

If a failure has occurred to the engine subsystem, the minutes_of_flight is stored in the variable old_minutes_of_flight for use in the next frame.

```
old_minutes_of_flight = minutes_of_flight;
```

See Appendix C for a complete source code listing.

3.7.3 Engine_status

The variable engine_status is a computed variable defining the current state of the engine.

3.7.3.1 Initialization

The variable engine_status is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
engine_status =        engine_stat_data[ 2];
```

3.7.3.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.3.2.1 Algorithm

If the engine_status is WORKING, the integrator_gain and engine_power are computed.

```
if (engine_status == WORKING)
{
    integrator_gain += gov_i_gain *
        (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
    if (integrator_gain > 0.5)
        integrator_gain = 0.5;
    else if (integrator_gain < -0.5)
        integrator_gain = -0.5;

    engine_power += integrator_gain;
}
else                        /* Damaged */
{
    integrator_gain = 0.0;
    if (engine_power > 0.7)
        engine_power = 0.7;
}
```

If the engine_status is WORKING, engine_speed is computed.

```
if (engine_status == WORKING)
    engine_speed += (engine_drive_torque - engine_load_torque)
        / POWERTRAIN_INERTIA;
```

If the engine_status is BROKEN, sound is halted.

```
if (engine_status == BROKEN)/* crippled condition */
{
    sound_stop_cont_sound (SOUND_OF_STOP_ENGINE,
SOUND_OF_VARY_ENGINE);
    sound_stop_cont_sound (SOUND_OF_STOP_ROTOR,
SOUND_OF_VARY_ROTOR);
    fuel_flow *= 50.0;  /* fuel leak */
}
```

If a failure has broken the engine subsystem, engine_status is set to BROKEN.

```
void    engine_break_engine ()
{
   engine_status = BROKEN;
   engine_speed = 0.0;
   number_of_engines = 1;
}
```

If the engine subsystem has been repaired, engine_status is set to WORKING.

```
void    engine_repair_engine ()
{
   engine_repair_engine_oil ();
   engine_status = WORKING;
   number_of_engines = 2;
}
```

See Appendix C for a complete source code listing.

3.7.4 Starting_engine

The variable starting_engine is a computed Boolean defining the current state of the engine in a starting mode.

3.7.4.1 Initialization

The variable starting_engine is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
starting_engine =       engine_stat_data[ 3];
```

3.7.4.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.4.2.1 Algorithm

If the engine is starting, engine_percent_torque is limited to .01 until it exceeds the delta, at which time the starting_engine flag is set to FALSE.

```
if (starting_engine)
{
    if (engine_percent_torque - .01 < .0001)        /* within a delta */
       starting_engine = FALSE;
    else
       engine_percent_torque = .01;
}
```

See Appendix C for a complete source code listing.

3.7.5 Number_of_engines

The variable starting_engine is a computed Boolean defining the current state of the engine in a starting mode.

3.7.5.1 Initialization

The variable starting_engine is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
number_of_engines =      engine_stat_data[ 4];
```

3.7.5.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.5.2.1 Algorithm

If the engine subsystem has been broken, the number_of_engines is reset to 1 by a call to CSC engine_break_engine.

```
void    engine_break_engine ()
{
   engine_status = BROKEN;
   engine_speed = 0.0;
   number_of_engines = 1;
}
```

If the engine subsystem has been repaired, the number_of_engines is reset to 2 by a call to CSC engine_repair_engine.

```
void    engine_repair_engine ()
{
    engine_repair_engine_oil ();
   engine_status = WORKING;
   number_of_engines = 2;
}
```

The number_of_engines is used to compute the engine_drive_torque.

```
    engine_drive_torque = engine_power * number_of_engines *
       (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
engine_speed);
```

The number_of_engines is used to compute the engine_percent_torque.

```
    engine_percent_torque = engine_drive_torque /
       (MAX_ENGINE_TORQUE * number_of_engines);
```

See Appendix C for a complete source code listing.

3.7.6 Engine_is_damaged

The variable engine_is_damaged is a computed Boolean defining the current state of the engine damage.

3.7.6.1 Initialization

The variable engine_is_damaged is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
engine_is_damaged =      engine_stat_data[ 5];
```

3.7.6.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.6.2.1 Algorithm

If engine oil is damaged, engine_is_damaged is set to TRUE by a call to CSC engine_damage_engine_oil.

```
void    engine_damage_engine_oil ()
{
#if DO_CFAIL
     controls_start_failure_lamp_flashing (MASTER_CAUTION);
     controls_start_failure_lamp_flashing (ENGINE_FAILURE);
#endif
   engine_is_damaged = TRUE;
}
```

If engine oil is repaired, engine_is_damaged is set to FALSE by a call to CSC engine_repair_engine_oil.

```
void    engine_repair_engine_oil ()
{
#if DO_CFAIL
     controls_failure_lamp_off (ENGINE_FAILURE);
   engine_is_damaged = FALSE;
#endif
}
```

See Appendix C for a complete source code listing.

3.7.7 Transmission_is_damaged

The variable transmission_is_damaged is a computed Boolean defining the current state of the transmission damage.

3.7.7.1 Initialization

The variable transmission_is_damaged is initialized during execution of the CSU engine_init, called by CSC rwa_init. Execution of the CSU engine_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
transmission_is_damaged = engine_stat_data[ 6];
```

3.7.7.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine_simul.

3.7.7.2.1 Algorithm

If engine transmission filter is damaged, transmission_is_damaged is set to TRUE by a call to CSC engine_damage_transmission_filter.

```
void    engine_damage_transmission_filter ()
{
#if DO_SFAIL
    controls_start_failure_lamp_flashing (MASTER_CAUTION);
    controls_start_failure_lamp_flashing (TRANSMISSION_FAILURE);
   transmission_is_damaged = TRUE;
#endif
}
```

If engine transmission filter is repaired, transmission_is_damaged is set to FALSE by a call to CSC engine_repair_transmission_filter.

```
void    engine_repair_transmission_filter ()
{
#if DO_SFAIL
    controls_failure_lamp_off (TRANSMISSION_FAILURE);
   transmission_is_damaged = FALSE;
#endif
}
```

See Appendix C for a complete source code listing.

3.8 Kinemat_data

This data array consists of kinematics constants and limits for the vehicle and its control.

3.8.1 GRAV_CONSTANT

GRAV_CONSTANT is a constant defining the gravitational constant.

3.8.1.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define GRAV_CONSTANT        kinemat_data[ 0]
```

3.8.1.2 Usage

During real-time execution, this constant is not recomputed.

3.8.1.2.1 Algorithm

The constant is used to compute g_force during execution of CSU veh_spec_kinematics_simul.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
```

See APPENDIX D for a complete source code listing.

3.8.2 SIN_AOA_LIMIT

SIN_AOA_LIMIT is a constant defining the sine of the angle_of_attack limit.

3.8.2.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define SIN_AOA_LIMIT        kinemat_data[ 1]
```

3.8.2.2 Usage

During real-time execution, this constant is not recomputed.

3.8.2.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

3.8.3 COS_AOA_LIMIT

COS_AOA_LIMIT is a constant defining the cosine of the angle_of_attack limit.

3.8.3.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define COS_AOA_LIMIT        kinemat_data[ 2]
```

3.8.3.2 Usage

During real-time execution, this constant is not recomputed.

3.8.3.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

3.8.4 SIN_YAW_LIMIT

SIN_YAW_LIMIT is a constant defining the sine of the yaw limit.

3.8.4.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define SIN_YAW_LIMIT        kinemat_data[ 3]
```

3.8.4.2 Usage

During real-time execution, this constant is not recomputed.

3.8.4.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

3.8.5 COS_YAW_LIMIT

COS_YAW_LIMIT is a constant defining the cosine of the yaw limit.

3.8.5.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define COS_YAW_LIMIT        kinemat_data[ 4]
```

3.8.5.2 Usage

During real-time execution, this constant is not recomputed.

3.8.5.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

3.8.6 DISPLAY_SPEED_LIMIT

DISPLAY_SPEED_LIMIT is a constant defining the lower limit of the displayed speed.

3.8.6.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define DISPLAY_SPEED_LIMIT   kinemat_data[ 5]
```

3.8.6.2 Usage

During real-time execution, this constant is not recomputed.

3.8.6.2.1 Algorithm

The constant is used to control computation of the velocity_pitch.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
   velocity_pitch = asin (vertical_speed);
else
   velocity_pitch = 0.0;
```

The constant is used to control computation of the normalized velocity vector.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

See APPENDIX D for a complete source code listing.

3.9 Kinemat_init_data

This data array consists of initial values for kinematics variables including velocity, angle-of-attack, pitch, altitude, heading, and g-force.

3,9.1 Pos_unit_vel

Pos_unit_vel is an array defining the positive unit velocity vector.

3.9.1.1 Initialization

The array is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
pos_unit_vel[Y] =    kinemat_init_data[ 1];
pos_unit_vel[Z] =    kinemat_init_data[ 2];
```

3.9.1.2 Usage

During real-time execution, this array is not recomputed.

3.9.1.2.1 Algorithm

The array is returned as the normalized velocity vector under certain conditions.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

See APPENDIX D for a complete source code listing.

3.9.2 Neg_unit_vel

Neg_unit_vel is an array defining the negative unit velocity vector.

3.9.2.1 Initialization

The array is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
neg_unit_vel[X] =   kinemat_init_data[ 3];
neg_unit_vel[Y] =   kinemat_init_data[ 4];
neg_unit_vel[Z] =   kinemat_init_data[ 5];
```

3.9.2.2 Usage

During real-time execution, this array is not recomputed.

3.9.2.2.1 Algorithm

The array is returned as the normalized velocity vector under certain conditions.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

See APPENDIX D for a complete source code listing.

3.9.3 Sin_aoa

Sin_aoa is a variable defining the sine of the angle-of-attack.

3.9.3.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
sin_aoa =          kinemat_init_data[ 6];
```

3.9.3.2 Usage

During real-time execution, this variable is recomputed.

3.9.3.2.1 Algorithm

The value of sin_aoa is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
   sin_aoa = -1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
   sin_aoa = 1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else
{
    sin_aoa = -norm_vel[Z];
    cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
                    norm_vel[Y]);
    sin_yaw = norm_vel[X] / cos_aoa;
    cos_yaw = norm_vel[Y] / cos_aoa;
}
```

Sin_aoa is used to compute a component of the velocity_to_body matrix.

```
velocity_to_body[1][2] = -sin_aoa;
```

See APPENDIX D for a complete source code listing.

3.9.4 Cos_aoa

Cos_aoa is a variable defining the cosine of the angle-of-attack.

3.9.4.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
cos_aoa =           kinemat_init_data[ 7];
```

3.9.4.2 Usage

During real-time execution, this variable is recomputed.

3.9.4.2.1 Algorithm

The value of cos_aoa is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
   sin_aoa = -1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
   sin_aoa = 1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else
{
    sin_aoa = -norm_vel[Z];
    cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
                    norm_vel[Y]);
   sin_yaw = norm_vel[X] / cos_aoa;
   cos_yaw = norm_vel[Y] / cos_aoa;
}
```

Cos_aoa is used to compute a components of the velocity_to_body matrix, roll and heading.

```
    temp = cos_aoa;


  velocity_to_body[1][0] = -velocity_to_body[0][1] * temp;
  velocity_to_body[1][1] =  velocity_to_body[0][0] * temp;


  temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
       body_to_world[1][1] * body_to_world[1][1]);
  if (temp < E_NANO)
  {
    roll = 0.0;
    heading = 0.0;
  }
  else
  {
      temp2 = (body_to_world[0][0] * body_to_world[1][1] -
            body_to_world[0][1] * body_to_world[1][0]) / temp;
      if (temp2 > 1.0) temp2 = 1.0;
      roll = acos (temp2);
    if (body_to_world[1][1] * body_to_world[2][0] -
         body_to_world[1][0] * body_to_world[2][1] < 0.0)
       roll = -roll;
    if (body_to_world[1][0] >= 0.0)
       heading = acos (body_to_world[1][1] / temp);
    else
       heading = acos (-body_to_world[1][1] / temp) + PI;
  }
```

See APPENDIX D for a complete source code listing.

3.9.5 Sin_yaw

Sin_yaw is a variable defining the sine of the yaw angle.

3.9.5.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
sin_yaw =          kinemat_init_data[ 8];
```

3.9.5.2 Usage

During real-time execution, this variable is recomputed.

3.9.5.2.1 Algorithm

The value of sin_yaw is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
   sin_aoa = -1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
   sin_aoa = 1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else
{
    sin_aoa = -norm_vel[Z];
         cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
norm_vel[Y]);
    sin_yaw = norm_vel[X] / cos_aoa;
    cos_yaw = norm_vel[Y] / cos_aoa;
}
```

The value of sin_yaw is used to compute the value of a component of the velocity_to_body matrix.

```
velocity_to_body[0][1] = -sin_yaw;
```

See APPENDIX D for a complete source code listing.

3.9.6 Cos_yaw

Cos_yaw is a variable defining the cosine of the yaw angle.

3.9.6.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
cos_yaw =        kinemat_init_data[ 9];
```

3.9.6.2 Usage

During real-time execution, this variable is recomputed.

3.9.6.2.1 Algorithm

The value of cos_yaw is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
   sin_aoa = -1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
   sin_aoa = 1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
```

```
else
{
    sin_aoa = -norm_vel[Z];
    cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
                    norm_vel[Y]);
    sin_yaw = norm_vel[X] / cos_aoa;
    cos_yaw = norm_vel[Y] / cos_aoa;
}
```

The value of cos_yaw is used to compute the value of a component of the velocity_to_body matrix.

```
velocity_to_body[0][0] = cos_yaw;
```

See APPENDIX D for a complete source code listing.

3.9.7 Altitude

Altitude is a variable defining the altitude above mean sea level, the database datum.

3.9.7.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
altitude =        kinemat_init_data[10];
```

3.9.7.2 Usage

During real-time execution, this variable is recomputed.

3.9.7.2.1 Algorithm

The value of altitude is set by assignment of the 'Z' component of the position vector.

```
altitude = position[Z];
```

If the value of altitude is negative, the altitude is limited to 0.0.

```
if (altitude < 0.0)
   altitude = 0.0;
```

See APPENDIX D for a complete source code listing.

3.9.8 Body_pitch

Body_pitch is a variable defining the angle of body pitch.

3.9.8.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
body_pitch =        kinemat_init_data[11];
```

3.9.8.2 Usage

During real-time execution, this variable is recomputed.

3.9.8.2.1 Algorithm

The value of body_pitch is computed as the arcsine of a component of the body_to_world matrix.

```
body_pitch = asin (body_to_world[1][2]);
```

External access to the body_pitch is achieved through a call to the CSC kinematics_get_body_pitch. The value return includes an offset constant that allows for the adjustment of the body_pitch reference.

```
REAL kinematics_get_body_pitch ()
{
   return (body_pitch + body_pitch_offset);
}
```

See APPENDIX D for a complete source code listing.

3.9.9 Body_pitch_offset

Body_pitch_offset is a constant defining the offset angle of body pitch. This offset allows for the adjustment of the body_pitch reference.

3.9.9.1 Initialization

The constant is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
body_pitch_offset =  kinemat_init_data[12];
```

3.9.9.2 Usage

During real-time execution, this constant is not recomputed.

See APPENDIX D for a complete source code listing.

3.9.10 Velocity_pitch

Velocity_pitch is a variable defining the cosine of the angle-of-attack.

3.9.10.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
velocity_pitch =     kinemat_init_data[13];
```

3.9.10.2 Usage

During real-time execution, this variable is recomputed.

3.9.10.2.1 Algorithm

The value of velocity_pitch is computed as the arcsine of the vertical_speed. If the true_airspeed is small, then the velocity_pitch is set to 0.0.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
   velocity_pitch = asin (vertical_speed);
else
   velocity_pitch = 0.0;
```

See APPENDIX D for a complete source code listing.

3.9.11 Roll

Roll is a variable defining the roll angle of the vehicle.

3.9.11.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
roll =          kinemat_init_data[14];
```

3.9.11.2 Usage

During real-time execution, this variable is recomputed.

3.9.11.2.1 Algorithm

The value of roll is computed from components of the body_to_world matrix.

```
temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
     body_to_world[1][1] * body_to_world[1][1]);


if (temp < E_NANO)
{
  roll = 0.0;
  heading = 0.0;
}
else
{
     temp2 = (body_to_world[0][0] * body_to_world[1][1] -
            body_to_world[0][1] * body_to_world[1][0]) / temp;
     if (temp2 > 1.0) temp2 = 1.0;
     roll = acos (temp2);
   if (body_to_world[1][1] * body_to_world[2][0] -
        body_to_world[1][0] * body_to_world[2][1] < 0.0)
      roll = -roll;
   if (body_to_world[1][0] >= 0.0)
      heading = acos (body_to_world[1][1] / temp);
   else
      heading = acos (-body_to_world[1][1] / temp) + PI;
}
```

See APPENDIX D for a complete source code listing.

3.9.12 Heading

Heading is a variable defining the heading angle the vehicle.

3.9.12.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
heading =        kinemat_init_data[15];
```

3.9.12.2 Usage

During real-time execution, this variable is recomputed.

3.9.12.2.1 Algorithm

The value of heading is computed from components of the body_to_world matrix.

```
temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
     body_to_world[1][1] * body_to_world[1][1]);


if (temp < E_NANO)
{
  roll = 0.0;
  heading = 0.0;
}
else
{
     temp2 = (body_to_world[0][0] * body_to_world[1][1] -
             body_to_world[0][1] * body_to_world[1][0]) / temp;
     if (temp2 > 1.0) temp2 = 1.0;
     roll = acos (temp2);
  if (body_to_world[1][1] * body_to_world[2][0] -
        body_to_world[1][0] * body_to_world[2][1] < 0.0)
      roll = -roll;
  if (body_to_world[1][0] >= 0.0)
     heading = acos (body_to_world[1][1] / temp);
  else
     heading = acos (-body_to_world[1][1] / temp) + PI;
}
```

See APPENDIX D for a complete source code listing.

3.9.13 True_airspeed

True_airspeed is a variable defining the true airspeed of the vehicle.

3.9.13.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
true_airspeed =     kinemat_init_data[16];
```

3.9.13.2 Usage

During real-time execution, this variable is recomputed.

3.9.13.2.1 Algorithm

The value of true_airspeed is computed from the velocity vector.

```
true_airspeed = sqrt (velocity[X] * velocity[X] + velocity[Y] * velocity[Y]
      + velocity[Z] * velocity[Z]);
```

True_airspeed is used to compute indicated_airspeed.

```
indicated_airspeed = true_airspeed * sqrt (air_density (altitude) /
      air_density(0.0));
```

True_airspeed is used to compute the normalized velocity vector.

```
if (true_airspeed < E_MILLI)
{
   norm_vel[X] = 0.0;
   norm_vel[Y] = 1.0;
   norm_vel[Z] = 0.0;
}
else
{
   norm_vel[X] = velocity[X] / true_airspeed;
   norm_vel[Y] = velocity[Y] / true_airspeed;
   norm_vel[Z] = velocity[Z] / true_airspeed;
}
```

True_airspeed is used to compute g_force.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
```

True_airspeed is used to control computation of velocity_pitch.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
   velocity_pitch = asin (vertical_speed);
else
   velocity_pitch = 0.0;
```

When access externally to the normalized velocity vector is requested, true_airspeed controls the value of the returned variable.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

See APPENDIX D for a complete source code listing.

3.9.14 Indicated_airspeed

Indicated_airspeed is a variable defining the indicated airspeed of the vehicle.

3.9.14.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
indicated_airspeed = kinemat_init_data[17];
```

3.9.14.2 Usage

During real-time execution, this variable is recomputed.

3.9.14.2.1 Algorithm

The value of indicated_airspeed is computed from the true_airspeed and corrected for altitude.

```
indicated_airspeed = true_airspeed * sqrt (air_density (altitude) /
      air_density(0.0));
```

See APPENDIX D for a complete source code listing.

3.9.15 G_force

G_force is a variable defining the "g" force exerted on the vehicle.

3.9.15.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
g_force =        kinemat_init_data[18];
```

3.9.15.2 Usage

During real-time execution, this variable is recomputed.

3.9.15.2.1 Algorithm

The value of g_force is set computed from the true_airspeed and angular velocity.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
```

See APPENDIX D for a complete source code listing.

3.9.16 Vertical_speed

Vertical_speed is a variable defining the vertical speed of the vehicle.

3.9.16.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
vertical_speed =    kinemat_init_data[19];
```

3.9.16.2 Usage

During real-time execution, this variable is recomputed.

3.9.16.2.1 Algorithm

The value of vertical_speed is computed from the normalized velocity vector and gravity, and adjusted using the true_airspeed.

```
vertical_speed = vec_dot_prod (norm_vel, gravity);


vertical_speed *= true_airspeed;
```

Vertical_speed is used to compute velocity_pitch.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
   velocity_pitch = asin (vertical_speed);
else
   velocity_pitch = 0.0;
```

See APPENDIX D for a complete source code listing.

3.9.17 Gravity

Gravity is a vector defining the gravity components of the vehicle.

3.9.17.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
gravity[X] =      kinemat_init_data[20];
gravity[Y] =      kinemat_init_data[21];
gravity[Z] =      kinemat_init_data[22];
```

3.9.17.2 Usage

During real-time execution, this variable is recomputed.

3.9.17.2.1 Algorithm

The value of gravity is assigned from the components of the body_to_world matrix.

```
gravity[X] = body_to_world[0][2];
gravity[Y] = body_to_world[1][2];
gravity[Z] = body_to_world[2][2];
```

The 'Z' component of the gravity vector is used to compute g_force.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
```

The value of vertical_speed is computed from the normalized velocity vector and gravity, and adjusted using the true_airspeed.

```
vertical_speed = vec_dot_prod (norm_vel, gravity);
```

See APPENDIX D for a complete source code listing.

3.9.18 Norm_vel

Norm_vel is a vector defining the normalized velocity vector.

3.9.18.1 Initialization

The variable is initialized during execution of the CSU veh_spec_kinematics_init, called by CSC rwa_init. Execution of the CSU veh_spec_kinematics_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
norm_vel[X] =      kinemat_init_data[23];
norm_vel[Y] =      kinemat_init_data[24];
norm_vel[Z] =      kinemat_init_data[25];
```

3.9.18.2 Usage

During real-time execution, this variable is recomputed.

3.9.18.2.1 Algorithm

The value of norm_vel vector is computed from the true_airspeed and velocity vector.

```
if (true_airspeed < E_MILLI)
{
   norm_vel[X] = 0.0;
   norm_vel[Y] = 1.0;
   norm_vel[Z] = 0.0;
}
else
{
   norm_vel[X] = velocity[X] / true_airspeed;
   norm_vel[Y] = velocity[Y] / true_airspeed;
   norm_vel[Z] = velocity[Z] / true_airspeed;
}
```

The norm_vel is used to compute and control computation of the sin_aoa, cos_aoa, sin_yaw, and cos_yaw.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
   sin_aoa = -1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
   sin_aoa = 1.0;
   cos_aoa = 0.0;
   sin_yaw = 0.0;
   cos_yaw = 1.0;
}
```

```
  else
  {
     sin_aoa = -norm_vel[Z];
          cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
norm_vel[Y]);
     sin_yaw = norm_vel[X] / cos_aoa;
     cos_yaw = norm_vel[Y] / cos_aoa;
  }
```

The value of vertical_speed is computed from the normalized velocity vector and gravity, and adjusted using the true_airspeed.

```
vertical_speed = vec_dot_prod (norm_vel, gravity);
```

See APPENDIX D for a complete source code listing.

3.10 Hellfr_miss_char

This data array consists of characteristics and parameters describing a Hellfire missile system and its performance constraints.

3.10.1 HELLFIRE_ARM_TIME

HELLFIRE_ARM_TIME is a constant defining the hellfire missile arm time delay before firing in ticks.

3.10.1.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define HELLFIRE_ARM_TIME        hellfr_miss_char[ 0]
```

3.10.1.2 Usage

During real-time execution, this variable is not recomputed.

3.10.1.2.1 Algorithm

HELLFIRE_ARM_TIME is used to control computation of the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
  ,else
      missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.2 HELLFIRE_BURNOUT_TIME

HELLFIRE_BURNOUT_TIME is a constant defining the time of powered flight for hellfire missile in ticks.

3.10.2.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define HELLFIRE_BURNOUT_TIME      hellfr_miss_char[ 1]
```

3.10.2.2 Usage

During real-time execution, this variable is not recomputed.

3.10.2.2.1 Algorithm

HELLFIRE_BURNOUT_TIME is used to control computation of the missile flyout speed by a call to the CSC missile_hellfire_fly.

```
  time = mptr->time;
/*/
 * Find the current missile speed . . .. The equations used are different
 * before and after motor burnout.
/*/
  if (time < HELLFIRE_BURNOUT_TIME)
  {
     mptr->speed = mptr->init_speed + (speed_factor *
      (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                              hellfire_burn_speed_coeff, time) ));
  }
  else
  {
     mptr->speed = mptr->init_speed + (speed_factor *
      (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                              hellfire_coast_speed_coeff, time) ));
  }
```

See APPENDIX G for a complete source code listing.

3.10.3 HELLFIRE_MAX_FLIGHT_TIME

HELLFIRE_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the hellfire missile assumed in ticks.

3.10.3.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define HELLFIRE_MAX_FLIGHT_TIME  hellfr_miss_char[ 2]
```

3.10.3.2 Usage

During real-time execution, this variable is not recomputed.

3.10.3.2.1 Algorithm

HELLFIRE_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual hellfire missile by a call to the CSU missile_hellfire_init.

```
mptr->max_flight_time = HELLFIRE_MAX_FLIGHT_TIME;
```

If not defined, the max_flight_time is set to the time-of-flight to the maximum range limit plus one by a call to the CSC missile_hellfire_fire.

```
#ifdef notdeff
  if( max_range_limit > 0.0 )
    mptr->max_flight_time =
       1.0 + missile_hellfire_calc_tof( max_range_limit );
#endif
```

See APPENDIX G for a complete source code listing.

3.10.4 SPEED_0

SPEED_0 is a constant defining the reference turn speed used to compute the ratio for the maximum turn angle.

3.10.4.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define SPEED_0              hellfr_miss_char[ 3]
```

3.10.4.2 Usage

During real-time execution, this variable is not recomputed.

3.10.4.2.1 Algorithm

SPEED_0 is used to compute the maximum turn angle for an individual hellfire missile by a call to the CSC missile_hellfire_fly.

```
mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / SPEED_0) * THETA_0);
```

See APPENDIX G for a complete source code listing.

3.10.5 THETA_0

THETA_0 is a constant defining the reference maximum turn angle which is scaled for speed.

3.10.5.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define THETA_0           hellfr_miss_char[ 4]
```

3.10.5.2 Usage

During real-time execution, this variable is not recomputed.

3.10.5.2.1 Algorithm

THETA_0 is used to compute the maximum turn angle for an individual hellfire missile by a call to the CSC missile_hellfire_fly.

```
mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / SPEED_0) * THETA_0);
```

See APPENDIX G for a complete source code listing.

3.10.6 SIN_UNGUIDE

SIN_UNGUIDE is a constant defining the sine of the delta pitch angle for an unguided hellfire missile

3.10.6.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_UNGUIDE            hellfr_miss_char[ 5]
```

3.10.6.2 Usage

During real-time execution, this variable is not recomputed.

3.10.6.2.1 Algorithm

SIN_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
       missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                           SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                           COS_LOCK, COS_TERM, COS_LOSE);
  else
       missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                           COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                           SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.7 COS_UNGUIDE

COS_UNGUIDE is a constant defining the cosine of the delta pitch angle for an unguided hellfire missile

3.10.7.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_UNGUIDE            hellfr_miss_char[ 6]
```

3.10.7.2 Usage

During real-time execution, this variable is not recomputed.

3.10.7.2.1 Algorithm

COS_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
  else
      missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.8 SIN_CLIMB

SIN_CLIMB is a constant defining the sine of the delta pitch angle for a climbing hellfire missile

3.10.8.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_CLIMB            hellfr_miss_char[ 7]
```

3.10.8.2 Usage

During real-time execution, this variable is not recomputed.

3.10.8.2.1 Algorithm

SIN_CLIMB is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
   else
      missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.9 COS_CLIMB

COS_CLIMB is a constant defining the cosine of the delta pitch angle for a climbing hellfire missile

3.10.9.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_CLIMB            hellfr_miss_char[ 8]
```

3.10.9.2 Usage

During real-time execution, this variable is not recomputed.

3.10.9.2.1 Algorithm

COS_CLIMB is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
  if( max_range_limit > 0  &&
     kinematics_range_squared (veh_kinematics, mptr->location) >
    max_range_squared )
     missile_target_ground( mptr );
  else if (time < HELLFIRE_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
  else
      missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.10 SIN_LOCK

SIN_LOCK is a constant defining the sine of the lock cone angle for a locked-on hellfire missile

3.10.10.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_LOCK              hellfr_miss_char[ 9]
```

3.10.10.2 Usage

During real-time execution, this variable is not recomputed.

3.10.10.2.1 Algorithm

SIN_LOCK is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
       missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                           SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                           COS_LOCK, COS_TERM, COS_LOSE);
  else
       missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                           COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                           SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.11 COS_LOCK

COS_LOCK is a constant defining the cosine of the lock cone angle for a locked-on hellfire missile

3.10.11.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOCK            hellfr_miss_char[10]
```

3.10.11.2 Usage

During real-time execution, this variable is not recomputed.

3.10.11.2.1 Algorithm

COS_LOCK is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
       missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                           SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                           COS_LOCK, COS_TERM, COS_LOSE);
   else
       missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                           COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                           SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.12 COS_TERM

COS_TERM is a constant defining the cosine of the terminal pitch angle for a locked-on hellfire missile

3.10.12.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_TERM              hellfr_miss_char[11]
```

3.10.12.2 Usage

During real-time execution, this variable is not recomputed.

3.10.12.2.1 Algorithm

COS_TERM is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
   else
      missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.10.13 COS_LOSE

COS_LOSE is a constant defining the cosine of the pitch angle for a loss-of-lock-on hellfire missile

3.10.13.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOSE            hellfr_miss_char[12]
```

3.10.13.2 Usage

During real-time execution, this variable is not recomputed.

3.10.13.2.1 Algorithm

COS_LOSE is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * If the missile is not armed, fly in a search trajectory; otherwise, fly
 * in a targeted trajectory.
/*/
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (time < HELLFIRE_ARM_TIME)
       missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                           SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                           COS_LOCK, COS_TERM, COS_LOSE);
   else
       missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                           COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                           SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

See APPENDIX G for a complete source code listing.

3.11 Hellfr_miss_poly_deg

The hellfr_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the time-of-flight, the burn speed, and the coast speed for the Hellfire missile.

3.11.1 HELLFIRE_TOF_DEG

HELLFIRE_TOF_DEG is a constant defining the polynomial degree for the hellfire missile time-of-flight coefficient data array. HELLFIRE_TOF_DEG is the first element of the hellfr_miss_poly_deg array.

3.11.1.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.11. - Hellfire Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define HELLFIRE_TOF_DEG        hellfr_miss_poly_deg[ 0]
```

3.11.1.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for HELLFIRE_TOF_DEG is 9, especially, the declared size of the hellfire_tof_coeff array is 10.

3.11.1.2.1 Algorithm

HELLFIRE_TOF_DEG is used to compute the hellfire missile time of flight using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_calc_tof. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
  time = missile_util_eval_poly( HELLFIRE_TOF_DEG,
                                  hellfire_tof_coeff, range );
 return( (time / speed_factor) );
```

See APPENDIX G for a complete source code listing.

3.11.2 HELLFIRE_BURN_SPEED_DEG

HELLFIRE_BURN_SPEED_DEG is a constant defining the polynomial degree for the hellfire missile burn speed coefficient data array. HELLFIRE_BURN_SPEED_DEG is the second element of the hellfr_miss_poly_deg array.

3.11.2.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.11. - Hellfire Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define HELLFIRE_BURN_SPEED_DEG    hellfr_miss_poly_deg[ 1]
```

3.11.2.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for HELLFIRE_BURN_SPEED_DEG is 9, especially, the declared size of the hellfire_burn_speed_coeff array is 10.

3.11.2.2.1 Algorithm

HELLFIRE_BURN_SPEED_DEG is used to compute the hellfire missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
mptr->time = 0.0;
vec_copy (launch_point, mptr->location);
mat_copy (launch_to_world, mptr->orientation);
mptr->speed = launch_speed +
    (speed_factor * (missile_util_eval_poly
                        (HELLFIRE_BURN_SPEED_DEG,
                          hellfire_burn_speed_coeff,
                        0.0) ));
mptr->init_speed = launch_speed;
```

HELLFIRE_BURN_SPEED_DEG is used to compute the hellfire missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
if (time < HELLFIRE_BURNOUT_TIME)
{
   mptr->speed = mptr->init_speed +
      (speed_factor *
         (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                        hellfire_burn_speed_coeff, time) ));
}
else
{
   mptr->speed = mptr->init_speed +
      (speed_factor *
         (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                        hellfire_coast_speed_coeff, time) ));
}
```

See APPENDIX G for a complete source code listing.

3.11.3 HELLFIRE_COAST_SPEED_DEG

HELLFIRE_COAST_SPEED_DEG is a constant defining the polynomial degree for the hellfire missile coast speed coefficient data array. HELLFIRE_COAST_SPEED_DEG is the third element of the hellfr_miss_poly_deg array.

3.11.3.1 Initialization

The constant is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.11. - Hellfire Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define HELLFIRE_COAST_SPEED_DEG  hellfr_miss_poly_deg[ 2]
```

3.11.3.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for HELLFIRE_COAST_SPEED_DEG is 9, especially, the declared size of the hellfire_coast_speed_coeff array is 10.

3.11.3.2.1 Algorithm

HELLFIRE_COAST_SPEED_DEG is used to compute the hellfire missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
if (time < HELLFIRE_BURNOUT_TIME)
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
       (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                     hellfire_burn_speed_coeff, time) ));
}
else
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
       (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                     hellfire_coast_speed_coeff, time) ));
}
```

See APPENDIX G for a complete source code listing.

3.12 Hellfire_tof_coeff

The hellfire_tof_coeff array consists of the coefficients for a polynomial equation defining the Hellfire missile time-of-flight with respect to range in the form using the Newton-Raphson method.

3.12.1 Initialization

The hellfire_tof_coeff array is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.12. - Hellfire Missile Time-Of-Flight Data Array for a summary of the constants data.

The array has a maximum size of 10 elements.

3.12.2 Usage

During real-time execution, this array is not recomputed. HELLFIRE_TOF_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.12.2.1 Algorithm

The hellfire_tof_coeff array is used to compute the hellfire missile time of flight using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_calc_tof. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and range.

```
time = missile_util_eval_poly( HELLFIRE_TOF_DEG,
                               hellfire_tof_coeff, range );
return( (time / speed_factor) );
```

See APPENDIX G for a complete source code listing.

3.13 Hellfire_burn_speed_coeff

The hellfire_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the Hellfire missile burn speed with respect to time in the form using the Newton-Raphson method.

3.13.1 Initialization

The hellfire_burn_speed_coeff array is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.13. - Hellfire Missile Burn Speed Data Array for a summary of the array data.

The array has a maximum size of 10 elements.

3.13.2 Usage

During real-time execution, this array is not recomputed. HELLFIRE_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.13.2.1 Algorithm

The hellfire_burn_speed_coeff array is used to compute the hellfire missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
mptr->time = 0.0;
vec_copy (launch_point, mptr->location);
mat_copy (launch_to_world, mptr->orientation);
mptr->speed = launch_speed +
    (speed_factor * (missile_util_eval_poly
                        (HELLFIRE_BURN_SPEED_DEG,
                          hellfire_burn_speed_coeff,
                        0.0) ));
mptr->init_speed = launch_speed;
```

The hellfire_burn_speed_coeff array is used to compute the hellfire missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
if (time < HELLFIRE_BURNOUT_TIME)
{
    mptr->speed = mptr->init_speed +
      (speed_factor *
        (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                    hellfire_burn_speed_coeff, time) ));
}
else
{
    mptr->speed = mptr->init_speed +
      (speed_factor *
        (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                    hellfire_coast_speed_coeff, time) ));
}
```

See APPENDIX G for a complete source code listing.

3.14 Hellfire_coast_speed_coeff

The hellfire_coast_speed_coeff array consists of the coefficients for a polynomial equation defining the Hellfire missile coast speed with respect to time in the form using the Newton-Raphson method.

3.14.1 Initialization

The hellfire_coast_speed_coeff array is initialized during execution of the CSU missile_hellfire_init, called by CSC weapons_init. Execution of the CSU missile_hellfire_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.14. - Hellfire Missile Coast Speed Data Array for a summary of the constants data.

The array has a maximum size of 10 elements.

3.14.2 Usage

During real-time execution, this array is not recomputed. HELLFIRE_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.14.2.1 Algorithm

The hellfire_coast_speed_coeff array is used to compute the hellfire missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_hellfire_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
if (time < HELLFIRE_BURNOUT_TIME)
{
   mptr->speed = mptr->init_speed +
      (speed_factor *
         (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                     hellfire_burn_speed_coeff, time) ));
}
else
{
   mptr->speed = mptr->init_speed +
      (speed_factor *
         (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                     hellfire_coast_speed_coeff, time) ));
}
```

See APPENDIX G for a complete source code listing.

3.15 Maverick_miss_char

The maverick_miss_char array consists of characteristics and parameters describing a Maverick missile system and its performance constraints.

3.15.1 MAVERICK_ARM_TIME

MAVERICK_ARM_TIME is a constant defining the maverick missile arm time delay before firing in ticks.

3.15.1.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_ARM_TIME          maverick_miss_char[ 0]
```

3.15.1.2 Usage

During real-time execution, this constant is not recomputed.

3.15.1.2.1 Algorithm

MAVERICK_ARM_TIME is used to control computation of the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly.  The missile ignores
 * any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
        missile_target_agm (mptr, NULL, SIN_UNGUIDE,
                              COS_UNGUIDE, SIN_CLIMB,
                              COS_CLIMB, SIN_LOCK,
                              COS_LOCK, COS_TERM, COS_LOSE);
   else
   {
       TObjectP object = mvptr -> object_being_tracked;
```

```
/*/
 *   Try to find a target.  If one is found, fly towards it in the
 *   proper trajectory, otherwise, fly in a search trajectory.
/*/
    if (object != NO_OBJECT)
    {
        VECTOR target_location;
        GetLocationOfTObject (object, target_location);
        mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                            COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                            SIN_LOCK, COS_LOCK,
                            COS_TERM, COS_LOSE);
    }
    else
    {
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                    mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                    TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                            SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                            COS_LOCK, COS_TERM, COS_LOSE);
    }
}
```

See APPENDIX I for a complete source code listing.

3.15.2 MAVERICK_BURNOUT_TIME

MAVERICK_BURNOUT_TIME is a constant defining the time of powered flight for maverick missile in ticks.

3.15.2.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_BURNOUT_TIME      maverick_miss_char[ 1]
```

3.15.2.2 Usage

During real-time execution, this constant is not recomputed.

3.15.2.2.1 Algorithm

MAVERICK_BURNOUT_TIME is used to control computation of the missile flyout speed by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the current missile speed and the cosine of the maximum
 *     allowed turn angle. The equations used are different before and
 *     after motor burnout.
/*/
   if (time < MAVERICK_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly
                        (MAVERICK_BURN_SPEED_DEG,
                        maverick_burn_speed_coeff, time) + mptr->init_speed;
  }
  else
  {
      mptr->speed = missile_util_eval_poly
                        (MAVERICK_COAST_SPEED_DEG,
                        maverick_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX I for a complete source code listing.

3.15.3 MAVERICK_MAX_FLIGHT_TIME

MAVERICK_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the maverick missile assumed in ticks.

3.15.3.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_MAX_FLIGHT_TIME   maverick_miss_char[ 2]
```

3.15.3.2 Usage

During real-time execution, this constant is not recomputed.

3.15.3.2.1 Algorithm

MAVERICK_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual maverick missile by a call to the CSU missile_maverick_init.

```
mptr->max_flight_time = MAVERICK_MAX_FLIGHT_TIME;
```

The max_flight_time for the each missile is set to the maximum flight time for an individual maverick missile by a call to the CSU missile_maverick_init.

```
for (i = 0; i < num_missiles; i++)
{
    maverick_array[i].mptr.state = MAVERICK_FREE;
    maverick_array[i].mptr.max_flight_time =
                                    MAVERICK_MAX_FLIGHT_TIME;
    maverick_array[i].mptr.max_turn_directions = 1;
    maverick_array[i].object_being_tracked = NO_OBJECT;
    maverick_array[i].sensor_id = NULL;
}
```

See APPENDIX I for a complete source code listing.

3.15.4 MAVERICK_LOCK_THRESHOLD

MAVERICK_LOCK_THRESHOLD is a constant defining the cosine squared of the lock threshold angle for the maverick missile.

3.15.4.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization

and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_LOCK_THRESHOLD   maverick_miss_char[ 3]
```

3.15.4.2 Usage

During real-time execution, this constant is not recomputed.

3.15.4.2.1 Algorithm

MAVERICK_LOCK_THRESHOLD is used to initialize the maximum cone threshold angle for the maverick missile by a call to the CSU missile_maverick_init.

```
    maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
```

The maverick_cone_of_threshold and detectibility are computed by a call to the CSC missile_maverick_detectibility.

```
detectibility = sign (dotProduct) * dotProduct * dotProduct /
            vec_dot_prod (to_target, to_target);

/* if the object is outside the detection cone of the sensor,
 * return a detectibility of 0.
 */

 if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
     != NULL)
{
     switch (mvptr -> mptr.state)
     {
      case MAVERICK_READY:
        maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
       break;
      case MAVERICK_FLYING:
        maverick_cone_threshold = MAVERICK_HOLD_THRESHOLD;
       break;
      case MAVERICK_FREE:
      default:
        printf ("MaverickDetectibility: Maverick not READY or FLYING\n");
```

```
        maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
        break;
    }

    if (detectibility < maverick_cone_threshold)
    detectibility = 0.0;
}
else
{
    printf ("MaverickDetectibility: no missile for sensorID %d\n",
            sensor_id);
}

return (detectibility);
```

See APPENDIX I for a complete source code listing.

3.15.5 MAVERICK_HOLD_THRESHOLD

MAVERICK_HOLD_THRESHOLD is a constant defining the cosine squared of the hold threshold angle for the maverick missile.

3.15.5.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_HOLD_THRESHOLD   maverick_miss_char[ 4]
```

3.15.5.2 Usage

During real-time execution, this constant is not recomputed.

3.15.5.2.1 Algorithm

MAVERICK_HOLD_THRESHOLD is used to compute the maverick_cone_of_threshold and detectibility by a call to the CSC missile_maverick_detectibility.

```
detectibility = sign (dotProduct) * dotProduct * dotProduct /
          vec_dot_prod (to_target, to_target);

/* if the object is outside the detection cone of the sensor,
 * return a detectibility of 0.
*/

 if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
    != NULL)
{
    switch (mvptr -> mptr.state)
    {
      case MAVERICK_READY:
        maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
       break;
      case MAVERICK_FLYING:
        maverick_cone_threshold = MAVERICK_HOLD_THRESHOLD;
       break;
      case MAVERICK_FREE:
      default:
        printf ("MaverickDetectibility: Maverick not READY or FLYING\n");
        maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
       break;
    }

    if (detectibility < maverick_cone_threshold)
    detectibility = 0.0;
}
else
{
    printf ("MaverickDetectibility: no missile for sensorID %d\n",
           sensor_id);
}

 return (detectibility);
```

See APPENDIX I for a complete source code listing.

3.15.6 SPEED_0

SPEED_0 is a constant defining the reference turn speed used to compute the ratio for the maximum turn angle.

3.15.6.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define SPEED_0            maverick_miss_char[ 5]
```

3.15.6.2 Usage

During real-time execution, this constant is not recomputed.

3.15.6.2.1 Algorithm

SPEED_0 is used to compute the maximum turn angle for an individual maverick missile by a call to the CSC missile_maverick_fly.

```
/*/
 * Note that this is a temporary method of finding turn angle.
/*/
   mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
          mptr->init_speed)) * THETA_0);
```

See APPENDIX I for a complete source code listing.

3.15.7 THETA_0

THETA_0 is a constant defining the reference maximum turn angle which is scaled for speed.

3.15.7.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define THETA_0              maverick_miss_char[ 6]
```

3.15.7.2 Usage

During real-time execution, this constant is not recomputed.

3.15.7.2.1 Algorithm

THETA_0 is used to compute the maximum turn angle for an individual maverick missile by a call to the CSC missile_maverick_fly.

```
/*/
 * Note that this is a temporary method of finding turn angle.
/*/
  mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
        mptr->init_speed)) * THETA_0);
```

See APPENDIX I for a complete source code listing.

3.15.8 SIN_UNGUIDE

SIN_UNGUIDE is a constant defining the sine of the delta pitch angle for an unguided maverick missile

3.15.8.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_UNGUIDE          maverick_miss_char[ 7]
```

3.15.8.2 Usage

During real-time execution, this constant is not recomputed.

3.15.8.2.1 Algorithm

SIN_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly.  The missile ignores
 * any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                            SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                            COS_LOCK, COS_TERM, COS_LOSE);
  else
  {
        TObjectP object = mvptr -> object_being_tracked;
/*/
 *,    Try to find a target.  If one is found, fly towards it in the
 *     proper trajectory, otherwise, fly in a search trajectory.
/*/
      if (object != NO_OBJECT)
      {
            VECTOR target_location;
            GetLocationOfTObject (object, target_location);
           mvptr->target_vehicle_id = object -> var.vehicleID;
           missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                               COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                               SIN_LOCK, COS_LOCK, COS_TERM,
                               COS_LOSE);
      }
      else
      {
           mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
           if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                       mptr -> orientation[1]) < 0)
               printf ("missile_maverick_fly: TrackAcquire: %s\n",
                       TrackErrString ());
           missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
               SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
             COS_LOSE);
      }
  }
```

See APPENDIX I for a complete source code listing.

3.15.9 COS_UNGUIDE

COS_UNGUIDE is a constant defining the cosine of the delta pitch angle for an unguided maverick missile

3.15.9.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_UNGUIDE          maverick_miss_char[ 8]
```

3,15.9.2 Usage

During real-time execution, this constant is not recomputed.

3.15.9.2.1 Algorithm

COS_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile_hellfire_fly.

```
/*/
 * Find the target point to which the missile is to fly. The missile ignores
 * any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
       missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                           SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                           COS_LOCK, COS_TERM, COS_LOSE);
   else
   {
        TObjectP object = mvptr -> object_being_tracked;
/*/
 *    Try to find a target. If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
      if (object != NO_OBJECT)
      {
           VECTOR target_location;
```

```
      GetLocationOfTObject (object, target_location);
      mvptr->target_vehicle_id = object -> var.vehicleID;
      missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM,
                          COS_LOSE);
  }
  else
  {
      mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
      if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                    mptr -> orientation[1]) < 0)
         printf ("missile_maverick_fly: TrackAcquire: %s\n",
                 TrackErrString ());
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
         SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
        COS_LOSE);
  }
}
```

See APPENDIX I for a complete source code listing.

3.15.10 SIN_CLIMB

SIN_CLIMB is a constant defining the sine of the delta pitch angle for a climbing maverick missile

3.15.10.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_CLIMB           maverick_miss_char[ 9]
```

3.15.10.2 Usage

During real-time execution, this constant is not recomputed.

3.15.10.2.1 Algorithm

SIN_CLIMB is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly.  The missile ignores
 * any targets until it is armed.
/*/
   if (time < MAVERICK_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
  else
  {
       TObjectP object = mvptr -> object_being_tracked;
/*/
 *    Try to find a target.  If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
    if (object != NO_OBJECT)
    {
         VECTOR target_location;
         GetLocationOfTObject (object, target_location);
        mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                            COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                            SIN_LOCK, COS_LOCK, COS_TERM,
                            COS_LOSE);
    }
    else
    {
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                   mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                    TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
           SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS_LOSE);
    }
  }
```

See APPENDIX I for a complete source code listing.

3.15.11 COS_CLIMB

COS_CLIMB is a constant defining the cosine of the delta pitch angle for a climbing maverick missile

3.15.11.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_CLIMB          maverick_miss_char[10]
```

3.15.11.2 Usage

During real-time execution, this constant is not recomputed.

3.15.11.2.1 Algorithm

COS_CLIMB is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly.  The missile ignores
 * any targets until it is armed.
/*/
   if (time < MAVERICK_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
  else
  {
      TObjectP object = mvptr -> object_being_tracked;
/*/
 *    Try to find a target.  If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
     if (object != NO_OBJECT)
     {
         VECTOR target_location;
```

```
        GetLocationOfTObject (object, target_location);
        mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                            COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                            SIN_LOCK, COS_LOCK, COS_TERM,
                            COS_LOSE);
    }
    else
    {
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                    mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                    TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
            SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS_LOSE);
    }
}
```

See APPENDIX I for a complete source code listing.

3.15.12 SIN_LOCK

SIN_LOCK is a constant defining the sine of the lock cone angle for a locked-on maverick missile

3.15.12.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_LOCK            maverick_miss_char[11]
```

3.15.12.2 Usage

During real-time execution, this constant is not recomputed.

3.15.12.2.1 Algorithm

SIN_LOCK is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly. The missile ignores
 * any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                            SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                            COS_LOCK, COS_TERM, COS_LOSE);
    else
    {
        TObjectP object = mvptr -> object_being_tracked;
/*/
 *,   Try to find a target. If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
        if (object != NO_OBJECT)
        {
            VECTOR target_location;
            GetLocationOfTObject (object, target_location);
            mvptr->target_vehicle_id = object -> var.vehicleID;
            missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                                COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                                SIN_LOCK, COS_LOCK, COS_TERM,
                                COS_LOSE);
        }
        else
        {
            mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
            if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                        mptr -> orientation[1]) < 0)
                printf ("missile_maverick_fly: TrackAcquire: %s\n",
                        TrackErrString ());
            missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
              COS_LOSE);
        }
    }
```

See APPENDIX I for a complete source code listing.

3.15.13 COS_LOCK

COS_LOCK is a constant defining the cosine of the lock cone angle for a locked-on maverick missile

3.15.13.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOCK            maverick_miss_char[12]
```

3.15.13.2 Usage

During real-time execution, this constant is not recomputed.

3.15.13.2.1 Algorithm

COS_LOCK is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly. The missile ignores
 * any targets until it is armed.
/*/
   if (time < MAVERICK_ARM_TIME)
      missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                          SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                          COS_LOCK, COS_TERM, COS_LOSE);
  else
  {
      TObjectP object = mvptr -> object_being_tracked;
/*/
 *    Try to find a target. If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
     if (object != NO_OBJECT)
     {
         VECTOR target_location;
```

```
        GetLocationOfTObject (object, target_location);
        mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                            COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                            SIN_LOCK, COS_LOCK, COS_TERM,
                            COS_LOSE);
    }
    else
    {
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                    mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                    TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
            SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS_LOSE);
    }
}
```

See APPENDIX I for a complete source code listing.

3.15.14 COS_TERM

COS_TERM is a constant defining the cosine of the terminal pitch angle for a locked-on maverick missile

3.15.14.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_TERM            maverick_miss_char[13]
```

3.15.14.2 Usage

During real-time execution, this constant is not recomputed.

3.15.14.2.1 Algorithm

COS_TERM is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly. The missile ignores
 * any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                            SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                            COS_LOCK, COS_TERM, COS_LOSE);
  else
  {
        TObjectP object = mvptr -> object_being_tracked;
/*/
 *,   Try to find a target. If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
     if (object != NO_OBJECT)
     {
          VECTOR target_location;
          GetLocationOfTObject (object, target_location);
         mvptr->target_vehicle_id = object -> var.vehicleID;
         missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                             COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                             SIN_LOCK, COS_LOCK, COS_TERM,
                             COS_LOSE);
     }
     else
     {
         mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
          if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                     mptr -> orientation[1]) < 0)
              printf ("missile_maverick_fly: TrackAcquire: %s\n",
                      TrackErrString ());
         missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
              SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
            COS_LOSE);
     }
  }
```

See APPENDIX I for a complete source code listing.

3.15.15 COS_LOSE

COS_LOSE is a constant defining the cosine of the pitch angle for a loss-of-lock-on maverick missile

3.15.15.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOSE            maverick_miss_char[14]
```

3.15.15.2 Usage

During real-time execution, this constant is not recomputed.

3.15.15.2.1 Algorithm

COS_LOSE is used to compute the missile flyout path by a call to the CSC missile_maverick_fly.

```
/*/
 * Find the target point to which the missile is to fly.  The missile ignores
 * any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
       missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                           SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                           COS_LOCK, COS_TERM, COS_LOSE);
   else
   {
        TObjectP object = mvptr -> object_being_tracked;
/*/
 *    Try to find a target.  If one is found, fly towards it in the
 *    proper trajectory, otherwise, fly in a search trajectory.
/*/
      if (object != NO_OBJECT)
      {
           VECTOR target_location;
```

```
        GetLocationOfTObject (object, target_location);
        mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                            COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                            SIN_LOCK, COS_LOCK, COS_TERM,
                            COS_LOSE);
    }
    else
    {
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                    mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                    TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
            SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS_LOSE);
    }
}
```

See APPENDIX I for a complete source code listing.

3.16 Maverick_miss_poly_deg

The maverick_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Maverick missile.

3.16.1 MAVERICK_BURN_SPEED_DEG

MAVERICK_BURN_SPEED_DEG is a constant defining the polynomial degree for the Maverick missile burn speed coefficient data array. MAVERICK_BURN_SPEED_DEG is the first element of the maverick_miss_poly_deg array.

3.16.1.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.16. - Maverick Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define MAVERICK_BURN_SPEED_DEG    maverick_miss_poly_deg[ 0]
```

3.16.1.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for MAVERICK_BURN_SPEED_DEG is 4, especially, the declared size of the maverick_burn_speed_coeff array is 5.

3.16.1.2.1 Algorithm

MAVERICK_BURN_SPEED_DEG is used to compute the maverick missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_maverick_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
, mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  mat_copy (launch_to_world, mptr->orientation);
  mptr->speed = missile_util_eval_poly (MAVERICK_BURN_SPEED_DEG,
       maverick_burn_speed_coeff, 0.0) + launch_speed;
  mptr->init_speed = launch_speed;
```

MAVERICK_BURN_SPEED_DEG is used to compute the maverick missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_maverick_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
  mptr = &(mvptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum
 *     allowed turn angle. The equations used are different before
 *     and after motor burnout.
/*/
  if (time < MAVERICK_BURNOUT_TIME)
  {
     mptr->speed = missile_util_eval_poly (
           MAVERICK_BURN_SPEED_DEG,
           maverick_burn_speed_coeff, time) + mptr->init_speed;
```

```
}
else
{
    mptr->speed = missile_util_eval_poly (
            MAVERICK_COAST_SPEED_DEG,
            maverick_coast_speed_coeff, time) + mptr->init_speed;
}
```

See APPENDIX I for a complete source code listing.

3.16.2 MAVERICK_COAST_SPEED_DEG

MAVERICK_COAST_SPEED_DEG is a constant defining the polynomial degree for the maverick missile coast speed coefficient data array. MAVERICK_COAST_SPEED_DEG is the second element of the maverick_miss_poly_deg array.

3.16.2.1 Initialization

The constant is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.16. - Maverick Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define MAVERICK_COAST_SPEED_DEG   maverick_miss_poly_deg[ 1]
```

3.16.2.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for MAVERICK_COAST_SPEED_DEG is 4, especially, the declared size of the maverick_coast_speed_coeff array is 5.

3.16.2.2.1 Algorithm

MAVERICK_COAST_SPEED_DEG is used to compute the maverick missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_maverick_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
  mptr = &(mvptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum
 *     allowed turn angle.  The equations used are different before
 *     and after motor burnout.
/*/
  if (time < MAVERICK_BURNOUT_TIME)
  {
     mptr->speed = missile_util_eval_poly (
            MAVERICK_BURN_SPEED_DEG,
            maverick_burn_speed_coeff, time) + mptr->init_speed;
  }
  else
  {
     mptr->speed = missile_util_eval_poly (
            MAVERICK_COAST_SPEED_DEG,
            maverick_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX I for a complete source code listing.

3.17 Maverick_burn_speed_coeff

The maverick_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the Maverick missile burn speed with respect to time in the form using the Newton-Raphson method.

3.17.1 Initialization

The maverick_burn_speed_coeff array is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.17. - Maverick Missile Burn Speed Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

3.17.2 Usage

During real-time execution, this array is not recomputed. MAVERICK_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.17.2.1 Algorithm

The maverick_burn_speed_coeff array is used to compute the maverick missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_maverick_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
mptr->time = 0.0;
vec_copy (launch_point, mptr->location);
mat_copy (launch_to_world, mptr->orientation);
mptr->speed = missile_util_eval_poly (MAVERICK_BURN_SPEED_DEG,
        maverick_burn_speed_coeff, 0.0) + launch_speed;
mptr->init_speed = launch_speed;
```

The maverick_burn_speed_coeff array is used to compute the maverick missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_maverick_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
  mptr = &(mvptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum
 *     allowed turn angle. The equations used are different before
 *     and after motor burnout.
/*/
  if (time < MAVERICK_BURNOUT_TIME)
  {
     mptr->speed = missile_util_eval_poly (
            MAVERICK_BURN_SPEED_DEG,
            maverick_burn_speed_coeff, time) + mptr->init_speed;
  }
  else
  {
     mptr->speed = missile_util_eval_poly (
            MAVERICK_COAST_SPEED_DEG,
            maverick_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX I for a complete source code listing.

3.18 Maverick_coast_speed_coeff

The maverick_coast_speed_coeff array consists of the coefficients for a polynomial equation defining the Maverick missile coast speed with respect to time in the form using the Newton-Raphson method.

3.18.1 Initialization

The maverick_coast_speed_coeff array is initialized during execution of the CSU missile_maverick_init, called by CSC weapons_init. Execution of the CSU missile_maverick_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.18. - Maverick Missile Coast Speed Data Array for a summary of the constants data.

The array has a maximum size of 5 elements.

3.18.2 Usage

During real-time execution, this array is not recomputed. MAVERICK_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.18.2.1 Algorithm

The maverick_coast_speed_coeff array is used to compute the maverick missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_maverick_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
  mptr = &(mvptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum
 *     allowed turn angle. The equations used are different before
 *     and after motor burnout.
/*/
   if (time < MAVERICK_BURNOUT_TIME)
  {
     mptr->speed = missile_util_eval_poly (
            MAVERICK_BURN_SPEED_DEG,
            maverick_burn_speed_coeff, time) + mptr->init_speed;
```

```
}
else
{
    mptr->speed = missile_util_eval_poly (
            MAVERICK_COAST_SPEED_DEG,
            maverick_coast_speed_coeff, time) + mptr->init_speed;
}
```

See APPENDIX I for a complete source code listing.

3.19 Stinger_miss_char

The stinger_miss_char array consists of characteristics and parameters describing a Stinger missile system and its performance constraints.

3.19.1 STINGER_BURNOUT_TIME

STINGER_BURNOUT_TIME is a constant defining the time of powered flight for stinger missile in ticks.

3.19.1.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define STINGER_BURNOUT_TIME      stinger_miss_char[ 0]
```

3.19.1.2 Usage

During real-time execution, this constant is not recomputed.

3.19.1.2.1 Algorithm

STINGER_BURNOUT_TIME is used to control computation of the missile flyout speed and the cosine of the maximum allowed turn by a call to the CSC missile_stinger_fly.

```
/*/
 * Find the current missile speed and the cosine of the maximum
 *      allowed turn angle. The equations used are different before and
 *      after motor burnout.
/*/
    if (time < STINGER_BURNOUT_TIME)
   {
        mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
             stinger_burn_speed_coeff, time) + mptr->init_speed;
   }
   else
   {
        mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
             stinger_coast_speed_coeff, time) + mptr->init_speed;
   }
/*/
 * Note that this is a temporary method of finding turn angle.
/*/
   mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
        mptr->init_speed)) * THETA_0);
/*/
 * Try to find a target. If one is found, fly towards it in the
 * proper trajectory, otherwise, fly in a straight line.
/*/
    target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
         veh_list, mptr->location, mptr->orientation[1],
         STINGER_LOCK_THRESHOLD);
   if( max_range_limit > 0  &&
      kinematics_range_squared (veh_kinematics, mptr->location) >
     max_range_squared )
      missile_target_ground( mptr );
   else if (target != NULL)
   {
      sptr->target_vehicle_id = target->vehicleID;
       if (time < STINGER_BURNOUT_TIME)
         missile_target_intercept_pre_burnout (mptr, target,
              sptr->stinger_burn_range_coeff, STINGER_BURNOUT_TIME,
             STINGER_BURN_SPEED_DEG + 1,
             sptr->stinger_coast_range_coeff,
             sptr->stinger_coast_range_2_coeff,
             STINGER_COAST_SPEED_DEG + 1);
```

```
    else
        missile_target_intercept (mptr, target,
            sptr->stinger_coast_range_coeff,
            sptr->stinger_coast_range_2_coeff,
            STINGER_COAST_SPEED_DEG + 1);
}
else
{
    sptr->target_vehicle_id.vehicle = vehicleIrrelevant;
    missile_target_unguided (mptr);
}
```

See APPENDIX K for a complete source code listing.

3.19.2 STINGER_MAX_FLIGHT_TIME

STINGER_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the stinger missile assumed in ticks.

3.19.2.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define STINGER_MAX_FLIGHT_TIME   stinger_miss_char[ 1]
```

3.19.2.2 Usage

During real-time execution, this constant is not recomputed.

3.19.2.2.1 Algorithm

STINGER_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual stinger missile by a call to the CSU missile_stinger_init.

```
for (i = 0; i < num_missiles; i++)
{
   stinger_array[i].mptr.state = STINGER_FREE;
   stinger_array[i].mptr.max_flight_time =
                                          STINGER_MAX_FLIGHT_TIME;
   stinger_array[i].mptr.max_turn_directions = 1;
}
```

See APPENDIX K for a complete source code listing.

3.19.3 STINGER_LOCK_THRESHOLD

STINGER_LOCK_THRESHOLD is a constant defining the cosine squared of the lock threshold angle for the stinger missile.

3.19.3.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define STINGER_LOCK_THRESHOLD   stinger_miss_char[ 2]
```

3.19.3.2 Usage

During real-time execution, this constant is not recomputed.

3.19.3.2.1 Algorithm

STINGER_LOCK_THRESHOLD is used to compute a target during pre-launch by a call to the CSU missile_stinger_pre_launch.

```
/*/
 * Try to find a target.
/*/
   target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
         veh_list, launch_point, launch_to_world[1],
       STINGER_LOCK_THRESHOLD);
```

STINGER_LOCK_THRESHOLD is used to compute the target by a call to the CSC missile_stinger_fly.

```
/*/
 * Try to find a target. If one is found, fly towards it in the
 * proper trajectory, otherwise, fly in a straight line.
/*/
    target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
          veh_list, mptr->location, mptr->orientation[1],
          STINGER_LOCK_THRESHOLD);
```

See APPENDIX K for a complete source code listing.

3.19.4 SPEED_0

SPEED_0 is a constant defining the reference turn speed used to compute the ratio for the maximum turn angle.

3.19.4.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define SPEED_0            stinger_miss_char[ 3]
```

3.19.4.2 Usage

During real-time execution, this constant is not recomputed.

3.19.4.2.1 Algorithm

SPEED_0 is used to compute the maximum turn angle for an individual stinger missile by a call to the CSC missile_stinger_fly.

```
/*/
 * Note that this is a temporary method of finding turn angle.
/*/
  mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
        mptr->init_speed)) * THETA_0);
```

See APPENDIX K for a complete source code listing.

3.19.5 THETA_0

default is 15.0 deg/sec
THETA_0 is a constant defining the reference maximum turn angle which is scaled for speed.

3.19.5.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define THETA_0              stinger_miss_char[ 4]
```

3.19.5.2 Usage

During real-time execution, this constant is not recomputed.

3.19.5.2.1 Algorithm

THETA_0 is used to compute the maximum turn angle for an individual stinger missile by a call to the CSC missile_stinger_fly.

```
/*/
 * Note that this is a temporary method of finding turn angle.
/*/
  mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
        mptr->init_speed)) * THETA_0);
```

See APPENDIX K for a complete source code listing.

3.19.6 INVEST_DIST_SQ

The INVEST_DIST_SQ is a constant defining the area at a maximum speed of less than 100 m/sec.

3.19.6.1 Initialization

The INVEST_DIST_SQ is initialized during execution of the CSU missile_stinger_init, called by CSU weapons_init. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constant.

```
#define INVEST_DIST_SQ          stinger_miss_char[5]
```

3.19.6.2 Usage

During real-time execution, this constant is not recomputed.

3,19.6.2.1 Algorithm

INVEST_DIST_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile_stinger_fly.

```
/*/
 *    If the missile successfully flew, process the proximity fuze.
/*/
     if (sptr->target_vehicle_id.vehicle == vehicleIrrelevant)
        missile_fuze_prox (mptr, MSL_TYPE_MISSILE,
                      PROX_FUZE_ON_ALL_VEH,
                      &(sptr->target_vehicle_id), &(sptr->pptr),
                      veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
    else
        missile_fuze_prox (mptr, MSL_TYPE_MISSILE,
                      PROX_FUZE_ON_ONE_VEH,
                      &(sptr->target_vehicle_id), &(sptr->pptr),
                      veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
```

See APPENDIX K for a complete source code listing.

3.19.7 FUZE_DIST_SQ

FUZE_DIST_SQ is a constant defining the square of the radius of the cylinder describing the flechettes fly in a cylinder with a radius of 20 meters and a length of 750 meters

3.19.7.1 Initialization

The FUZE_DIST_SQ is initialized during execution of the CSU missile_stinger_init, called by CSU weapons_init. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constant.

```
#define FUZE_DIST_SQ          stinger_miss_char[ 6]
```

3.19.7.2 Usage

During real-time execution, this constant is not recomputed.

3.19.7.2.1 Algorithm

FUZE_DIST_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile_stinger_fly.

```
/*/
 *    If the missile successfully flew, process the proximity fuze.
/*/
      if (sptr->target_vehicle_id.vehicle == vehicleIrrelevant)
         missile_fuze_prox (mptr, MSL_TYPE_MISSILE,
                       PROX_FUZE_ON_ALL_VEH,
                       &(sptr->target_vehicle_id), &(sptr->pptr),
                       veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
     else
         missile_fuze_prox (mptr, MSL_TYPE_MISSILE,
                       PROX_FUZE_ON_ONE_VEH,
                       &(sptr->target_vehicle_id), &(sptr->pptr),
                       veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
```

See APPENDIX K for a complete source code listing.

3.20 Stinger_miss_poly_deg

The stinger_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Stinger missile.

3.20.1 STINGER_BURN_SPEED_DEG

STINGER_BURN_SPEED_DEG is a constant defining the polynomial degree for the Stinger missile burn speed coefficient data array.

STINGER_BURN_SPEED_DEG is the first element of the stinger_miss_poly_deg array. STINGER_BURN_SPEED_DEG is also known as stinger_miss_poly_deg[0].

3.20.1.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.20. - Stinger Missile Polynomial Degree Data Array for a summary of the constants data.

3.20.1.2 Usage

During real-time execution, this variable is not recomputed. The value for stinger_miss_poly_deg[0] is 1, especially, the declared size of the stinger_burn_speed_coeff array is 2. This value cannot be changed with a change to the source code because of other dependencies in the code structure.

3,20.1.2.1 Algorithm

STINGER_BURN_SPEED_DEG is used to compute the stinger missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_stinger_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Get a pointer to the generic elements of the STINGER missile. This
 * improves code readability.
/*/
  mptr = &(sptr->mptr);
/*/
 * Set the initial time, location, orientation and speed of the generic
 * missile.
/*/
  mptr->time = 0.0;
   vec_copy (launch_point, mptr->location);
   mat_copy (launch_to_world, mptr->orientation);
  mptr->speed = launch_speed +
    (speed_factor *
       missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
                     stinger_burn_speed_coeff, 0.0));
   mptr->init_speed = launch_speed;
```

STINGER_BURN_SPEED_DEG is used to compute the stinger missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_stinger_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
/*/
  mptr = &(sptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum allowed
 * turn angle. The equations used are different before and after
 * motor burnout.
/*/
  if (time < STINGER_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
           stinger_burn_speed_coeff, time) + mptr->init_speed;
  }
  else
  {
      mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
           stinger_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX K for a complete source code listing.

3.20.2 STINGER_COAST_SPEED_DEG

STINGER_COAST_SPEED_DEG is a constant defining the polynomial degree for the stinger missile coast speed coefficient data array. STINGER_COAST_SPEED_DEG is the second element of the stinger_miss_poly_deg array. STINGER_COAST_SPEED_DEG is also known as stinger_miss_poly_deg[1].

3.20.2.1 Initialization

The constant is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.20. - Stinger Missile Polynomial Degree Data Array for a summary of the constants data.

3.20.2.2 Usage

During real-time execution, this variable is not recomputed. The value for STINGER_COAST_SPEED_DEG is 3, especially, the declared size of the stinger_coast_speed_coeff array is 4. This value cannot be changed with a change to the source code because of other dependencies in the code structure.

3.20.2.2.1 Algorithm

STINGER_COAST_SPEED_DEG is used to compute the stinger missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_stinger_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
/*/
  mptr = &(sptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum allowed
 * turn angle. The equations used are different before and after
 * motor burnout.
/*/
  if (time < STINGER_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
          stinger_burn_speed_coeff, time) + mptr->init_speed;
  }
  else
  {
      mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
          stinger_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX K for a complete source code listing.

3.21 Stinger_burn_speed_coeff

The stinger_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the Stinger missile burn speed with respect to time in the form using the Newton-Raphson method.

3.21.1 Initialization

The stinger_burn_speed_coeff array is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.21. - Stinger Missile Burn Speed Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

3.21.2 Usage

During real-time execution, this array is not recomputed. STINGER_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.21.2.1 Algorithm

The stinger_burn_speed_coeff array is used to compute the stinger missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_stinger_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Get a pointer to the generic elements of the STINGER missile. This
 * improves code readability.
/*/
   mptr = &(sptr->mptr);
/*/
 * Set the initial time, location, orientation and speed of the generic
 * missile.
/*/
   mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (launch_to_world, mptr->orientation);
```

```
mptr->speed = launch_speed +
  (speed_factor *
    missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
                stinger_burn_speed_coeff, 0.0));
mptr->init_speed = launch_speed;
```

The stinger_burn_speed_coeff array is used to compute the stinger missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_stinger_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
/*/
  mptr = &(sptr->mptr);
  time = mptr->time;
/*/
 * Find the current missile speed and the cosine of the maximum allowed
 * turn angle. The equations used are different before and after
 * motor burnout.
/*/
  if (time < STINGER_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
          stinger_burn_speed_coeff, time) + mptr->init_speed;
  }
  else
  {
      mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
          stinger_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX K for a complete source code listing.

3.22 Stinger_coast_speed_coeff

The stinger_coast_speed_coeff array consists of the coefficients for a polynomial equation defining the Stinger missile coast speed with respect to time in the form using the Newton-Raphson method.

3.22.1 Initialization

The stinger_coast_speed_coeff array is initialized during execution of the CSU missile_stinger_init, called by CSC weapons_init. Execution of the CSU missile_stinger_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.22. - Stinger Missile Coast Speed Data Array for a summary of the constants data.

The array has a maximum size of 5 elements.

3.22.2 Usage

During real-time execution, this array is not recomputed. STINGER_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.22.2.1 Algorithm

The stinger_coast_speed_coeff array is used to compute the stinger missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_stinger_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 *  Set _mptr_ and _time_.  These values are created mostly for increased
 *  readablity.
/*/
  mptr = &(sptr->mptr);
  time = mptr->time;
/*/
 *  Find the current missile speed and the cosine of the maximum allowed
 *  turn angle.  The equations used are different before and after
 *  motor burnout.
/*/
   if (time < STINGER_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
           stinger_burn_speed_coeff, time) + mptr->init_speed;
  }
```

```
else
{
    mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
        stinger_coast_speed_coeff, time) + mptr->init_speed;
}
```

See APPENDIX K for a complete source code listing.

3.23 Tow_miss_char

The tow_miss_char array consists of characteristics and parameters describing a TOW missile system and its performance constraints.

3.23.1 TOW_BURNOUT_TIME

TOW_BURNOUT_TIME is a constant defining the time of powered flight for tow missile in ticks.

3.23.1.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.23. - Tow Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_BURNOUT_TIME      tow_miss_char[ 0]
```

3.23.1.2 Usage

During real-time execution, this constant is not recomputed.

3.23.1.2.1 Algorithm

TOW_BURNOUT_TIME is used to control computation of the missile flyout speed and the cosines of the maximum allowed turn angles in each direction by a call to the CSC missile_tow_fly.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *     allowed turn angles in each direction. The equations used are
 *     different before and after motor burnout.
/*/
```

```
if (time < TOW_BURNOUT_TIME)
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
        missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                    tow_burn_speed_coeff, time));
     missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}
else
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
        missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                    tow_coast_speed_coeff, time));
     missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX L for a complete source code listing.

3.23.2 TOW_RANGE_LIMIT_TIME

TOW_RANGE_LIMIT_TIME is a constant defining the range limit time for the tow missile in ticks; at this point the wire is cut, but the missile is allowed to fly to the maximum flight time.

3.23.2.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.23. - Tow Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_RANGE_LIMIT_TIME   tow_miss_char[ 1]
```

3.23.2.2 Usage

During real-time execution, this constant is not recomputed.

3.23.2.2.1 Algorithm

TOW_RANGE_LIMIT_TIME is used to control the wire cut for an individual tow missile by a call to the CSU missile_tow_fly

```
/*/
 * If the missile has reached its maximum range (not the maximum distance
 * its allowed to fly), cut the wire.
/*/
#ifdef notdeff
    if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)
      tptr->wire_is_cut = TRUE;
#endif
  , if (!tptr->wire_is_cut  &&
       ((time > TOW_RANGE_LIMIT_TIME)  ||
        (max_range_limit > 0  &&
          kinematics_range_squared (veh_kinematics, mptr->location)  >
        max_range_squared) ))
       tptr->wire_is_cut = TRUE;
```

See APPENDIX L for a complete source code listing.

3.23.3 TOW_MAX_FLIGHT_TIME

TOW_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the tow missile assumed in ticks; cosine of the max turn is greater than 1.0 beyond this point.

3.23.3.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.23. - Tow Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_MAX_FLIGHT_TIME   tow_miss_char[ 2]
```

3.23.3.2 Usage

During real-time execution, this constant is not recomputed.

3.23.3.2.1 Algorithm

TOW_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual tow missile by a call to the CSU missile_tow_init.

```
tptr->mptr.max_flight_time = TOW_MAX_FLIGHT_TIME;
```

See APPENDIX L for a complete source code listing.

3.24 Tow_miss_poly_deg

The tow_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the TOW missile.

3.24.1 TOW_BURN_SPEED_DEG

TOW_BURN_SPEED_DEG is a constant defining the polynomial degree for the tow missile burn speed coefficient data array.

3.24.1.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by the CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define TOW_BURN_SPEED_DEG   tow_miss_poly_deg[0]
```

3.24.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW_BURN_SPEED_DEG is 4, especially, the declared size of the tow_burn_speed_coeff is 5.

3.24.1.2.1 Algorithm

TOW_BURN_SPEED_DEG is used to compute the tow missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_tow_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
  mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  mat_copy (loc_sight_to_world, mptr->orientation);
  mptr->speed = launch_speed +
      (speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                              tow_burn_speed_coeff, 0.0));
  mptr->init_speed = launch_speed;
```

TOW_BURN_SPEED_DEG is used to compute the tow missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_tow_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *     allowed turn angles in each direction. The equations used are
 *     different before and after motor burnout.
/*/
  if (time < TOW_BURNOUT_TIME)
  {
     mptr->speed = mptr->init_speed +
       (speed_factor *
          missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                          tow_burn_speed_coeff, time));
       missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
```

```
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                      tow_coast_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX L for a complete source code listing.

3.24.2 TOW_COAST_SPEED_DEG

TOW_COAST_SPEED_DEG is a constant defining the polynomial degree for tow missile coast speed coefficient data array

3.24.2.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by the CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_SPEED_DEG  tow_miss_poly_deg[1]
```

3.24.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW_COAST_SPEED_DEG is 4, especially, the declared size of the tow_burn_speed_coeff is 5.

3.24.2.2.1 Algorithm

TOW_COAST_SPEED_DEG is used to compute the tow missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_tow_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *      allowed turn angles in each direction. The equations used are
 *      different before and after motor burnout.
/*/
```

```
if (time < TOW_BURNOUT_TIME)
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
        missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                    tow_burn_speed_coeff, time));
     missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}
else
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
        missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                    tow_coast_speed_coeff, time));
     missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX L for a complete source code listing.

3.24.3 TOW_BURN_TURN_DEG

TOW_BURN_TURN_DEG is a constant defining the polynomial degree for each tow missile burn turn coefficient data sub-array of the tow missile burn turn coefficient data array structure

3.24.3.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the constant data.

```
#define TOW_BURN_TURN_DEG    tow_miss_poly_deg[2]
```

3.24.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW_BURN_TURN_DEG is 1, especially, the declared size of the tow_burn_turn_coeff is 2. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
/*/
 * Coefficients for the cosine of max turn polynomials before motor burnout.
 * The structure _MAX_COS_COEFF_ is used to store the values for the turn
 * sideways, up, and down polynomials along with their order.
/*/

static MAX_COS_COEFF tow_burn_turn_coeff =
{
  1,            /* Order of the polynomials. */
  {
                /* Sidewards turn. */
    0.999976868652, /* a_0 - cos(rad)/tick */
   -3.5933955e-7    /* a_1 - cos(rad)/tick**2 */
  },
  {
                /* Upwards turn. */
    0.999960667258, /* a_0 - cos(rad)/tick */
   -3.1492328e-6    /* a_1 - cos(rad)/tick**2 */
  },
  {
                /* Downwards turn. */
    0.999978909989, /* a_0 - cos(rad)/tick */
   -7.8194991e-9    /* a_1 - cos(rad)/tick**2 */
  }
};
```

3.24.3.2.1 Algorithm

TOW_BURN_TURN_DEG is hard coded by type definition of MAX_COS_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during powered flight [burn] using the CSU missile_util_cos_coeff, and called by the CSU missile_tow_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *     allowed turn angles in each direction. The equations used are
 *     different before and after motor burnout.
/*/
```

```
if (time < TOW_BURNOUT_TIME)
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
        missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                    tow_burn_speed_coeff, time));
     missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}
else
{
   mptr->speed = mptr->init_speed +
     (speed_factor *
        missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                    tow_coast_speed_coeff, time));
     missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX L for a complete source code listing.

3.24.4 TOW_COAST_TURN_DEG

TOW_COAST_TURN_DEG is a constant defining the polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array structure

3.24.4.1 Initialization

The constant is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_TURN_DEG   tow_miss_poly_deg[3]
```

3.24.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value of each axis for TOW_COAST_TURN_DEG is 3, especially, the declared size of the tow_coast_turn_coeff is 4. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
/*/
 *  Coefficients for the cosine of max turn polynomials after motor burnout.
/*/

static MAX_COS_COEFF tow_coast_turn_coeff =
{
  3,              /* Order of the polynomials. */
  {
                  /* Sidewards turn. */
    0.99995112518,  /* a_0 - cos(rad)/tick */
    8.96333e-7,     /* a_1 - cos(rad)/tick**2 */
    -5.995375e-9,   /* a_2 - cos(rad)/tick**3 */
    1.162225e-11    /* a_3 - cos(rad)/tick**4 */
  },
  {
                  /* Upwards turn. */
    0.9998498495,   /* a_0 - cos(rad)/tick */
    1.657779e-6,    /* a_1 - cos(rad)/tick**2 */
    -8.231861e-9,   /* a_2 - cos(rad)/tick**3 */
    1.381832e-11    /* a_3 - cos(rad)/tick**4 */
  },
  {
                  /* Downwards turn. */
    0.9999714014,   /* a_0 - cos(rad)/tick */
    3.382077e-7,    /* a_1 - cos(rad)/tick**2 */
    -1.601259e-9,   /* a_2 - cos(rad)/tick**3 */
    2.623014e-12    /* a_3 - cos(rad)/tick**4 */
  }
};
```

3.24.4.2.1 Algorithm

TOW_COAST_TURN_DEG is hard coded by type definition of MAX_COS_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during unpowered flight [coast] using the CSU missile_util_cos_coeff, and called by the CSU missile_tow_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 *  Find the current missile speed and the cosines of the maximum
 *      allowed turn angles in each direction.  The equations used are
 *      different before and after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
   {
      mptr->speed = mptr->init_speed +
        (speed_factor *
           missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                        tow_burn_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
   }
   else
   {
      mptr->speed = mptr->init_speed +
        (speed_factor *
           missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                        tow_coast_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
   }
```

See APPENDIX L for a complete source code listing.

3.25 Tow_burn_speed_coeff

The tow_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the TOW missile burn speed with respect to time in the form using the Newton-Raphson method.

3.25.1 Initialization

The tow_burn_speed_coeff array is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.25. - TOW Missile Burn Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

3.25.2 Usage

During real-time execution, this array is not recomputed. TOW_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.25.2.1 Algorithm

Tow_burn_speed_coeff is used to compute the tow missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_tow_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
  mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  mat_copy (loc_sight_to_world, mptr->orientation);
  mptr->speed = launch_speed +
      (speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                              tow_burn_speed_coeff, 0.0));
  mptr->init_speed = launch_speed;
```

Tow_burn_speed_coeff is used to compute the tow missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_tow_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *     allowed turn angles in each direction. The equations used are
 *     different before and after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
  {
     mptr->speed = mptr->init_speed +
       (speed_factor *
          missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                         tow_burn_speed_coeff, time));
       missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
     mptr->speed = mptr->init_speed +
       (speed_factor *
```

```
            missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                          tow_coast_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
    }
```

See APPENDIX L for a complete source code listing.

3.26 Tow_coast_speed_coeff

This data array consists of the coefficients for a polynomial equation defining the TOW missile coast speed with respect to time in the form using the Newton-Raphson method.

3.26.1 Initialization

The tow_coast_speed_coeff array is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.26. - TOW Missile Coast Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

3.26.2 Usage

During real-time execution, this array is not recomputed. TOW_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.26.2.1 Algorithm

Tow_coast_turn_coeff is used to compute the tow missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_tow_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 *  Find the current missile speed and the cosines of the maximum
 *      allowed turn angles in each direction.  The equations used are
 *      different before and after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
    {
```

```
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                  tow_burn_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}
else
{
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                  tow_coast_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX L for a complete source code listing.

3.27 Tow_burn_turn_coeff

The tow_burn_turn_coeff two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the TOW missile maximum cosine of turn while powered with respect to time in the form using the Newton-Raphson method.

3.27.1 Initialization

The tow_burn_turn_coeff array is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.27. - TOW Missile Burn Turn Coefficients Data Array for a summary of the array data.

The array has a maximum size of 3 by 2 elements.

```
/*/
 * Coefficients for the cosine of max turn polynomials before motor burnout.
 * The structure _MAX_COS_COEFF_ is used to store the values for the turn
 * sideways, up, and down polynomials along with their order.
/*/

static MAX_COS_COEFF tow_burn_turn_coeff =
{
   1,              /* Order of the polynomials. */
```

```
  {
            /* Sidewards turn. */
    0.999976868652,  /* a_0 - cos(rad)/tick */
    -3.5933955e-7    /* a_1 - cos(rad)/tick**2 */
  },
  {
            /* Upwards turn. */
    0.999960667258,  /* a_0 - cos(rad)/tick */
    -3.1492328e-6    /* a_1 - cos(rad)/tick**2 */
  },
  {
            /* Downwards turn. */
    0.999978909989,  /* a_0 - cos(rad)/tick */
    -7.8194991e-9    /* a_1 - cos(rad)/tick**2 */
  }
};
```

Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

3.27.2 Usage

During real-time execution, this array is not recomputed. The size of the array in the type definition for MAX_COS_COEFF determines the number of elements of the array to be used in the polynomial evaluation.

3.27.2.1 Algorithm

Tow_burn_turn_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during powered flight [burn] using the CSU missile_util_cos_coeff, and called by the CSU missile_tow_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *      allowed turn angles in each direction. The equations used are
 *      different before and after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
   {
      mptr->speed = mptr->init_speed +
         (speed_factor *
```

```
            missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                        tow_burn_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
    }
    else
    {
      mptr->speed = mptr->init_speed +
        (speed_factor *
          missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                        tow_coast_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
    }
```

See APPENDIX L for a complete source code listing.

3.28 Tow_coast_turn_coeff

The tow_coast_turn_coeff two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the TOW missile maximum cosine of turn while unpowered with respect to time in the form using the Newton-Raphson method.

3.28.1 Initialization

The tow_coast_turn_coeff array is initialized during execution of the CSU missile_tow_init, called by CSC weapons_init. Execution of the CSU missile_tow_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.28. - TOW Missile Coast Turn Coefficients Data Array for a summary of the array data.

The array has a maximum size of 3 by 4 elements.

```
/*/
 * Coefficients for the cosine of max turn polynomials after motor burnout.
/*/

static MAX_COS_COEFF tow_coast_turn_coeff =
{
  3,              /* Order of the polynomials. */
  {
                /* Sidewards turn. */
    0.99995112518,  /* a_0 - cos(rad)/tick */
    8.96333e-7,     /* a_1 - cos(rad)/tick**2 */
```

```
    -5.995375e-9,    /* a_2 - cos(rad)/tick**3 */
    1.162225e-11     /* a_3 - cos(rad)/tick**4 */
  },
  {
            /* Upwards turn. */
    0.9998498495,    /* a_0 - cos(rad)/tick */
    1.657779e-6,     /* a_1 - cos(rad)/tick**2 */
    -8.231861e-9,    /* a_2 - cos(rad)/tick**3 */
    1.381832e-11     /* a_3 - cos(rad)/tick**4 */
  },
  {
            /* Downwards turn. */
    0.9999714014,    /* a_0 - cos(rad)/tick */
    3.382077e-7,     /* a_1 - cos(rad)/tick**2 */
    -1.601259e-9,    /* a_2 - cos(rad)/tick**3 */
    2.623014e-12     /* a_3 - cos(rad)/tick**4 */
  }
};
```

Changing the size of the array requires a recompile because of the hard coded multi-dimension characteristic.

3.28.2 Usage

During real-time execution, this array is not recomputed. The size of the array in the type definition for MAX_COS_COEFF determines the number of elements of the array to be used in the polynomial evaluation.

3.28.2.1 Algorithm

Tow_coast_turn_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during unpowered flight [coast] using the CSU missile_util_cos_coeff, and called by the CSU missile_tow_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *      allowed turn angles in each direction. The equations used are
 *      different before and after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
    {
```

```
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                     tow_burn_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}
else
{
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                     tow_coast_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX L for a complete source code listing.

3.29 Adat_miss_char

The adat_miss_char array consists of characteristics and parameters describing an ADAT missile system and its performance constraints.

3.29.1 ADAT_BURNOUT_TIME

ADAT_BURNOUT_TIME is a constant defining the time of powered flight for the adat missile in ticks.

3.29.1.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define ADAT_BURNOUT_TIME      adat_miss_char[ 0]
```

3.29.1.2 Usage

During real-time execution, this constant is not recomputed.

3.29.1.2.1 Algorithm

ADAT_BURNOUT_TIME is used to control computation of the missile flyout speed by a call to the CSC missile_adat_fly.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *  allowed turn angles in each direction.  The equations used are different
 *  before and after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
       mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
           adat_burn_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly (
            ADAT_BURN_TURN_DEG, adat_burn_turn_coeff, time);
  }
  else
  {
       mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
           adat_coast_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly (
            ADAT_COAST_TURN_DEG, adat_coast_turn_coeff, time);
  }
```

See APPENDIX E for a complete source code listing.

3.29.2 ADAT_MAX_FLIGHT_TIME

ADAT_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the adat missile in ticks.

3.29.2.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define ADAT_MAX_FLIGHT_TIME   adat_miss_char[ 1]
```

3.29.2.2 Usage

During real-time execution, this constant is not recomputed.

3.29.2.2.1 Algorithm

ADAT_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual adat missile by a call to the CSU missile_adat_init.

```
for (i = 0; i < num_missiles; i++)
{
   adat_array[i].mptr.state = ADAT_FREE;
     adat_array[i].mptr.max_flight_time = ADAT_MAX_FLIGHT_TIME;
    adat_array[i].mptr.max_turn_directions = 1;
}
```

See APPENDIX E for a complete source code listing.

3.29.3 INVEST_DIST_SQ

INVEST_DIST_SQ is a constant defining the area at a maximum speed of less than 100 meters/second.

3.29.3.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define INVEST_DIST_SQ        adat_miss_char[ 2]
```

3.29.3.2 Usage

During real-time execution, this constant is not recomputed.

3.29.3.2.1 Algorithm

INVEST_DIST_SQ is used to compute detonation of the proximity fuze by a call to the CSU missile_fuze_prox in the CSU missle_adat_fly.

```
/*/
 *    If the missile successfully flew, process the proximity fuze.
/*/
      missile_fuze_prox (mptr, MSL_TYPE_MISSILE, aptr->target_flag,
            &(aptr->target_vehicle_id), &(aptr->pptr), veh_list,
            INVEST_DIST_SQ, aptr->fuze_dist_sq);
```

See APPENDIX E for a complete source code listing.

3.29.4 HELO_FUZE_DIST_SQ

HELO_FUZE_DIST_SQ is a constant defining the square of the radius of the cylinder describing the proximity fuze area for a target setting of type HELO.

3.29.4.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define HELO_FUZE_DIST_SQ     adat_miss_char[ 3]
```

3.29.4.2 Usage

During real-time execution, this constant is not recomputed.

3.29.4.2.1 Algorithm

HELO_FUZE_DIST_SQ is used to compute the fuze distance for the missile target setting by a call to the CSC missile_adat_fire.

```
/*/
 * Set fuze distance and fuze target according to missile target
 * setting. Set network variables.
/*/
   switch (target_type)
   {
    case ADAT_TGT_GND:
       aptr->fuze_dist_sq = 0.0;
        aptr->target_flag = PROX_FUZE_ON_NO_VEH;
```

```
    break;
  case ADAT_TGT_HELO:
      aptr->fuze_dist_sq = HELO_FUZE_DIST_SQ;
      if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
          aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
    else
          aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
    break;
  case ADAT_TGT_AIR:
      aptr->fuze_dist_sq = AIR_FUZE_DIST_SQ;
      if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
          aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
    else
          aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
    break;
  default:
     aptr->fuze_dist_sq = 0.0;
      aptr->target_flag = PROX_FUZE_ON_NO_VEH;
      printf ("MISS_ADAT: Unknown target type %d\n", target_type);
    break;
  }
```

The fuze_dist_sq is used to compute the proximity fuze by a call to the CSU missile_fuze_prox in the CSU missile_adat_fly.

```
/*/
 *    If the missile successfully flew, process the proximity fuze.
/*/
      missile_fuze_prox (mptr, MSL_TYPE_MISSILE, aptr->target_flag,
            &(aptr->target_vehicle_id), &(aptr->pptr), veh_list,
            INVEST_DIST_SQ, aptr->fuze_dist_sq);
```

See APPENDIX E for a complete source code listing.

3.29.5 AIR_FUZE_DIST_SQ

AIR_FUZE_DIST_SQ is a constant defining the square of the radius of the cylinder describing the proximity fuze area for a target setting of type AIR.

3.29.5.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is

normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define AIR_FUZE_DIST_SQ      adat_miss_char[ 4]
```

3.29.5.2 Usage

During real-time execution, this constant is not recomputed.

3.29.5.2.1 Algorithm

AIR_FUZE_DIST_SQ is used to compute the fuze distance for the missile target setting by a call to the CSC missile_adat_fire.

```
/*/
 * Set fuze distance and fuze target according to missile target
 *  setting.  Set network variables.
/*/
  switch (target_type)
  {
   case ADAT_TGT_GND:
     aptr->fuze_dist_sq = 0.0;
      aptr->target_flag = PROX_FUZE_ON_NO_VEH;
    break;
   case ADAT_TGT_HELO:
     aptr->fuze_dist_sq = HELO_FUZE_DIST_SQ;
     if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
        aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
    else
        aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
    break;
   case ADAT_TGT_AIR:
     aptr->fuze_dist_sq = AIR_FUZE_DIST_SQ;
     if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
        aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
    else
        aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
    break;
```

```
  default:
     aptr->fuze_dist_sq = 0.0;
      aptr->target_flag = PROX_FUZE_ON_NO_VEH;
      printf ("MISS_ADAT: Unknown target type %d\n", target_type);
    break;
  }
```

The fuze_dist_sq is used to compute the proximity fuze by a call to the CSU missile_fuze_prox in the CSU missile_adat_fly.

```
/*/
 *    If the missile successfully flew, process the proximity fuze.
/*/
      missile_fuze_prox (mptr, MSL_TYPE_MISSILE, aptr->target_flag,
            &(aptr->target_vehicle_id), &(aptr->pptr), veh_list,
            INVEST_DIST_SQ, aptr->fuze_dist_sq);
```

See APPENDIX E for a complete source code listing.

3.29.6 ADAT_TEMP_BIAS_TIME

ADAT_TEMP_BIAS_TIME is a constant defining the time of temporal bias for the adat missile in ticks.

3.29.6.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define ADAT_TEMP_BIAS_TIME    adat_miss_char[ 5]
```

3.29.6.2 Usage

During real-time execution, this constant is not recomputed.

3.29.6.2.1 Algorithm

ADAT_TEMP_BIAS_TIME is used to compute the bias for the adat missile by a call to the CSC missile_adat_fly.

```
/*/
 * Find the target point, etc.
/*/
  if ((mptr->state == ADAT_GUIDE) || (mptr->state == ADAT_CLOSE))
  {
      if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state ==
                                                    ADAT_GUIDE))
    {
          bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
              adat_temp_bias_coeff, time);
        if (((tube / 2) * 2) == tube)
            missile_target_los_bias (mptr, sight_location,
                loc_sight_to_world, -bias, bias);
        else
            missile_target_los_bias (mptr, sight_location,
                loc_sight_to_world, bias, bias);
    }
    else
          missile_target_los (mptr, sight_location, loc_sight_to_world);
  }
  else if (mptr->state == ADAT_UNGUIDE)
      missile_target_unguided (mptr);
  else
      printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
```

See APPENDIX E for a complete source code listing.

3.29.7 CLOSE_RANGE

CLOSE_RANGE

3.29.7.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define CLOSE_RANGE        adat_miss_char[ 6]
```

3.29.7.2 Usage

During real-time execution, this constant is not recomputed.

3.29.7.2.1 Algorithm

CLOSE_RANGE is used to control the initial orientation of the adat missile by a call to the CSC missile_adat_fire.

```
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
   mptr->time = 0.0;
   vec_copy (launch_point, mptr->location);
   if (range_to_intercept < CLOSE_RANGE)
       mat_copy (loc_sight_to_world, mptr->orientation);
   else
   {
     if (((tube / 2) * 2) == tube)
          mat_mat_mul (tube_C_sight_left, loc_sight_to_world,
            mptr->orientation);
     else
          mat_mat_mul(tube_C_sight_right, loc_sight_to_world,
            mptr->orientation);
   }
   mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
        adat_burn_speed_coeff, 0.0) + launch_speed;
   mptr->init_speed = launch_speed;
```

CLOSE_RANGE is used to control the state of guidance for the adat missile by a call to the CSC missile_adat_fire.

```
/*/
 * If all was successful, put any flying missiles in an unguided state
 * and put this missile in a guided state.
/*/
   for (i = 0; i < num_adats; i++)
   {
     if ((adat_array[i].mptr.state == ADAT_GUIDE) ||
          (adat_array[i].mptr.state == ADAT_CLOSE))
        adat_array[i].mptr.state = ADAT_UNGUIDE;
   }
```

```
if (range_to_intercept < CLOSE_RANGE)
  mptr->state = ADAT_CLOSE;
else
  mptr->state = ADAT_GUIDE;
```

See APPENDIX E for a complete source code listing.

3.30 Adat_miss_poly_deg

The adat_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, maximum cosines of turns while unpowered, and temporal bias for the ADAT missile.

3.30.1 ADAT_BURN_SPEED_DEG

ADAT_BURN_SPEED_DEG is a constant defining the polynomial degree for ADAT missile burn speed coefficient data array. ADAT_BURN_SPEED_DEG is the first element of the adat_miss_poly_deg.

3.30.1.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_BURN_SPEED_DEG    adat_miss_poly_deg[ 0]
```

3.30.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT_BURN_SPEED_DEG is 9, especially, the declared size of the adat_burn_speed_coeff is 10.

3.30.1.2.1 Algorithm

ADAT_BURN_SPEED_DEG is used to compute the ADAT missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
  mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  if (range_to_intercept < CLOSE_RANGE)
      mat_copy (loc_sight_to_world, mptr->orientation);
  else
  {
    if (((tube / 2) * 2) == tube)
           mat_mat_mul (tube_C_sight_left, loc_sight_to_world,
             mptr->orientation);
    else
           mat_mat_mul(tube_C_sight_right, loc_sight_to_world,
             mptr->orientation);
  }
   mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
         adat_burn_speed_coeff, 0.0) + launch_speed;
  mptr->init_speed = launch_speed;
```

ADAT_BURN_SPEED_DEG is used to compute the ADAT missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
       mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
           adat_burn_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly(
                              ADAT_BURN_TURN_DEG,
                              adat_burn_turn_coeff, time);
  }
  else
  {
```

```
    mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
        adat_coast_speed_coeff, time) + mptr->init_speed;
    mptr->cos_max_turn[0] = missile_util_eval_poly(
                        ADAT_COAST_TURN_DEG,
                        adat_coast_turn_coeff, time);
}
```

See APPENDIX E for a complete source code listing.

3.30.2 ADAT_COAST_SPEED_DEG

ADAT_COAST_SPEED_DEG is a constant defining the polynomial degree for adat missile coast speed coefficient data array. ADAT_COAST_SPEED_DEG is the second element of the adat_miss_poly_deg.

3.30.2.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_COAST_SPEED_DEG  adat_miss_poly_deg[ 1]
```

3.30.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT_COAST_SPEED_DEG is 9, especially, the declared size of the adat_coast_speed_coeff is 10.

3.30.2.2.1 Algorithm

ADAT_COAST_SPEED_DEG is used to compute the ADAT missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
       mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
           adat_burn_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly(
                              ADAT_BURN_TURN_DEG,
                              adat_burn_turn_coeff, time);
  }
  else
  {
       mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
           adat_coast_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly(
                              ADAT_COAST_TURN_DEG,
                              adat_coast_turn_coeff, time);
  }
```

See APPENDIX E for a complete source code listing.

3.30.3 ADAT_BURN_TURN_DEG

ADAT_BURN_TURN_DEG is a constant defining the polynomial degree for the adat missile maximum cosine of turn angle, burn turn coefficient data array. ADAT_BURN_TURN_DEG is the third element of the adat_miss_poly_deg.

3.30.3.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_BURN_TURN_DEG     adat_miss_poly_deg[ 2]
```

3.30.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT_BURN_TURN_DEG is 9, especially, the declared size of the adat_burn_turn_coeff is 10.

3.30.3.2.1 Algorithm

ADAT_BURN_TURN_DEG is used to compute cosine of the maximum allowed turn angle for the ADAT missile during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 *, after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
       mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
            adat_burn_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly(
                              ADAT_BURN_TURN_DEG,
                              adat_burn_turn_coeff, time);
  }
  else
  {
       mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
            adat_coast_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly(
                              ADAT_COAST_TURN_DEG,
                              adat_coast_turn_coeff, time);
  }
```

See APPENDIX E for a complete source code listing.

3.30.4 ADAT_COAST_TURN_DEG

ADAT_COAST_TURN_DEG is a constant defining the polynomial degree for the adat missile maximum cosine of turn angle, coast turn coefficient data

array. ADAT_COAST_TURN_DEG is the fourth element of the adat_miss_poly_deg.

3.30.4.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_COAST_TURN_DEG    adat_miss_poly_deg[ 3]
```

3.30.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT_COAST_TURN_DEG is 9, especially, the declared size of the adat_coast_turn_coeff is 10.

3.30.4.2.1 Algorithm

ADAT_COAST_TURN_DEG is used to compute the cosine of the maximum allowed turn angle for the ADAT missile during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
          adat_burn_speed_coeff, time) + mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly(
                         ADAT_BURN_TURN_DEG,
                         adat_burn_turn_coeff, time);
  }
  else
  {
```

```
        mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
            adat_coast_speed_coeff, time) + mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly(
                            ADAT_COAST_TURN_DEG,
                            adat_coast_turn_coeff, time);
    }
```

See APPENDIX E for a complete source code listing.

3.30.5 ADAT_TEMP_BIAS_DEG

ADAT_TEMP_BIAS_DEG is a constant defining the polynomial degree for the adat missile temporal bias coefficient data array

3.30.5.1 Initialization

The constant is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_TEMP_BIAS_DEG     adat_miss_poly_deg[ 4]
```

3.30.5.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT_TEMP_BIAS_DEG is 9, especially, the declared size of the adat_coast_turn_coeff is 10.

3.30.5.2.1 Algorithm

ADAT_TEMP_BIAS_DEG is used to compute the temporal bias applied to the target location using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the target point, etc.
/*/
  if ((mptr->state == ADAT_GUIDE) || (mptr->state == ADAT_CLOSE))
  {
```

```
    if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state ==
                                                   ADAT_GUIDE))
   {
        bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
            adat_temp_bias_coeff, time);
     if (((tube / 2) * 2) == tube)
          missile_target_los_bias (mptr, sight_location,
              loc_sight_to_world, -bias, bias);
     else
          missile_target_los_bias (mptr, sight_location,
              loc_sight_to_world, bias, bias);
   }
   else
        missile_target_los (mptr, sight_location, loc_sight_to_world);
 }
 else if (mptr->state == ADAT_UNGUIDE)
    missile_target_unguided (mptr);
 else
    printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
```

See APPENDIX E for a complete source code listing.

3.31 Adat_burn_speed_coeff

The adat_burn_speed array consists of the coefficients for a polynomial equation defining the ADAT missile burn speed with respect to time in the form using the Newton-Raphson method.

3.31.1 Initialization

The adat_burn_speed array is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.31. - ADAT Missile Burn Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial before motor burnout.
/*/

static REAL adat_burn_speed_coeff[10] =
{
```

```
  2.296,          /* a_0 - m/tick */
  0.72990856,       /* a_1 - m/tick**2 */
  0.013310932,       /* a_2 - m/tick**3 */
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 10 elements.

3.31.2 Usage

During real-time execution, this array is not recomputed. ADAT_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.31.2.1 Algorithm

The adat_burn_speed_coeff array is used to initialize the tube to sight transformation matrices by a call to the CSU missile_adat_init.

```
/*/
 * Initialize the tube to sight transformation matrices.
/*/
   mag = sqrt (adat_burn_speed_coeff[0] * adat_burn_speed_coeff[0] +
        2.0 * adat_temp_bias_coeff[0] * adat_temp_bias_coeff[0]);
   tube_C_sight_right[1][0] = adat_temp_bias_coeff[0] / mag;
   tube_C_sight_right[1][1] = adat_burn_speed_coeff[0] / mag;
   tube_C_sight_right[1][2] = adat_temp_bias_coeff[0] / mag;
   mag = sqrt (tube_C_sight_right[1][0] * tube_C_sight_right[1][0] +
        tube_C_sight_right[1][1] * tube_C_sight_right[1][1]);
   tube_C_sight_right[0][0] = tube_C_sight_right[1][1] / mag;
   tube_C_sight_right[0][1] = -tube_C_sight_right[1][0] / mag;
   tube_C_sight_right[0][2] = 0.0;
   tube_C_sight_right[2][0] = tube_C_sight_right[1][2] *
        tube_C_sight_right[0][1];
   tube_C_sight_right[2][1] = -tube_C_sight_right[1][2] *
        tube_C_sight_right[0][0];
   tube_C_sight_right[2][2] = mag;
   mat_copy (tube_C_sight_right, tube_C_sight_left);
   tube_C_sight_left[0][1] = -tube_C_sight_left[0][1];
```

```
tube_C_sight_left[1][0] = -tube_C_sight_left[1][0];
tube_C_sight_left[2][0] = -tube_C_sight_left[2][0];
```

The adat_burn_speed_coeff array is used to compute the initial speed at launch of the ADAT missile using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
    adat_burn_speed_coeff, 0.0) + launch_speed;
```

The adat_burn_speed_coeff array is used to compute the ADAT missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
turn
 * angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
    if (time < ADAT_BURNOUT_TIME)
   {
       mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
           adat_burn_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly (
                                  ADAT_BURN_TURN_DEG,
                                  adat_burn_turn_coeff, time);
   }
   else
   {
       mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
           adat_coast_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly (
                                  ADAT_COAST_TURN_DEG,
                                  adat_coast_turn_coeff, time);
   }
```

See APPENDIX E for a complete source code listing.

3.32 Adat_coast_speed_coeff

This data array consists of the coefficients for a polynomial equation defining the ADAT missile coast speed with respect to time in the form using the Newton-Raphson method.

3.32.1 Initialization

The adat_coast_speed array is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.32. - ADAT Missile Coast Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial after motor burnout.
/*/

static REAL adat_coast_speed_coeff[10] =
{
 105.52162,          /* a_0 - m/tick */
  -1.0157285,        /* a_1 - m/tick**2 */
   5.6124330e-3,     /* a_2 - m/tick**3 */
  -1.6262608e-5,     /* a_3 - m/tick**4 */
   1.8991982e-8,     /* a_4 - m/tick**5 */
   0.0,
   0.0,
   0.0,
   0.0,
   0.0
};
```

The array has a maximum size of 10 elements.

3.32.2 Usage

During real-time execution, this array is not recomputed. ADAT_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.32.2.1 Algorithm

The adat_coast_speed_coeff array is used to compute the ADAT missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
turn
 * angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
          adat_burn_speed_coeff, time) + mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
                                 ADAT_BURN_TURN_DEG,
                                 adat_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
          adat_coast_speed_coeff, time) + mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
                                 ADAT_COAST_TURN_DEG,
                                 adat_coast_turn_coeff, time);
  }
```

See APPENDIX E for a complete source code listing.

3.33 Adat_burn_turn_coeff

The adat_burn_turn_coeff array consists of the coefficients for a polynomial equation defining the ADAT missile maximum cosine of turn while powered with respect to time in the form using the Newton-Raphson method.

3.33.1 Initialization

The adat_burn_turn array is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and

is performed sequentially. See TABLE 5.1.33. - ADAT Missile Burn Turn Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the cosine of max turn polynomial before motor burnout.
/*/

static REAL adat_burn_turn_coeff[10] =
{
  0.999993,       /* a_0 - cos(rad)/tick */
 -6.2386917e-7,   /* a_1 - cos(rad)/tick**2 */
  1.6146426e-7,   /* a_2 - cos(rad)/tick**3 */
 -9.720142e-7,    /* a_3 - cos(rad)/tick**4 */
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 10 elements.

3.33.2 Usage

During real-time execution, this array is not recomputed. ADAT_BURN_TURN_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.33.2.1 Algorithm

The adat_burn_turn_coeff array is used to compute the cosine of the maximum allowed turn angle of the ADAT missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
turn
```

```
 * angles in each direction.  The equations used are different before and
 * after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
          adat_burn_speed_coeff, time) + mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
                              ADAT_BURN_TURN_DEG,
                              adat_burn_turn_coeff,  time);
  }
  else
  {
      mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
          adat_coast_speed_coeff, time) + mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
                              ADAT_COAST_TURN_DEG,
                              adat_coast_turn_coeff,  time);
  }
```

See APPENDIX E for a complete source code listing.

3.34 Adat_coast_turn_coeff

This data array consists of the coefficients for a polynomial equation defining the ADAT missile maximum cosine of turn while unpowered with respect to time in the form using the Newton-Raphson method.

3.34.1 Initialization

The adat_coast_turn array is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.34. - ADAT Missile Coast Turn Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the cosine of max turn polynomial after motor burnout.
/*/

static REAL adat_coast_turn_coeff[10] =
{
```

```
  0.99753111,        /* a_0 - cos(rad)/tick */
  5.5817986e-5,       /* a_1 - cos(rad)/tick**2 */
  -5.1276276e-7,      /* a_2 - cos(rad)/tick**3 */
  2.2388593e-9,       /* a_3 - cos(rad)/tick**4 */
  -5.1964622e-12,     /* a_4 - cos(rad)/tick**5 */
  4.5499104e-15,      /* a_5 - cos(rad)/tick**6 */
  0.0,
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 10 elements.

3.34.2 Usage

During real-time execution, this array is not recomputed. ADAT_COAST_TURN_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.34.2.1 Algorithm

The adat_coast_turn_coeff array is used to compute the cosine of the maximum allowed turn angle of the ADAT missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
turn
 * angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < ADAT_BURNOUT_TIME)
   {
       mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
           adat_burn_speed_coeff, time) + mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly (
                                   ADAT_BURN_TURN_DEG,
                                   adat_burn_turn_coeff, time);
   }
   else
```

```
{
    mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
        adat_coast_speed_coeff, time) + mptr->init_speed;
    mptr->cos_max_turn[0] = missile_util_eval_poly (
                              ADAT_COAST_TURN_DEG,
                              adat_coast_turn_coeff, time);
}
```

See APPENDIX E for a complete source code listing.

3.35 Adat_temp_bias_coeff

The adat_temp_bias_coeff array consists of the coefficients for a polynomial equation defining the ADAT missile temporal bias with respect to time in the form using the Newton-Raphson method.

3.35.1 Initialization

The adat_temp_bias_coeff array is initialized during execution of the CSU missile_adat_init, called by CSC weapons_init. Execution of the CSU missile_adat_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.35. - ADAT Missile Temporal Bias Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the temporial bias polynomial.
/*/

static REAL adat_temp_bias_coeff[10] =
{
   5.3105657e-2,    /* a_0 - m */
   7.1795817e-2,    /* a_1 - m/tick */
   1.8084646e-2,    /* a_2 - m/tick**2 */
  -6.0083762e-4,    /* a_3 - m/tick**3 */
   4.6761091e-6,    /* a_4 - m/tick**4 */
   0.0,
   0.0,
   0.0,
   0.0,
   0.0
};
```

The array has a maximum size of 10 elements.

3.35.2 Usage

During real-time execution, this array is not recomputed. ADAT_TEMP_BIAS_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.35.2.1 Algorithm

The adat_temp_bias_coeff array is used to initialize the tube to sight transformation matrices by a call to the CSU missile_adat_init.

```
/*/
 * Initialize the tube to sight transformation matrices.
/*/
   mag = sqrt (adat_burn_speed_coeff[0] * adat_burn_speed_coeff[0] +
         2.0 * adat_temp_bias_coeff[0] * adat_temp_bias_coeff[0]);
   tube_C_sight_right[1][0] = adat_temp_bias_coeff[0] / mag;
   tube_C_sight_right[1][1] = adat_burn_speed_coeff[0] / mag;
   tube_C_sight_right[1][2] = adat_temp_bias_coeff[0] / mag;
   mag = sqrt (tube_C_sight_right[1][0] * tube_C_sight_right[1][0] +
         tube_C_sight_right[1][1] * tube_C_sight_right[1][1]);
   tube_C_sight_right[0][0] = tube_C_sight_right[1][1] / mag;
   tube_C_sight_right[0][1] = -tube_C_sight_right[1][0] / mag;
   tube_C_sight_right[0][2] = 0.0;
   tube_C_sight_right[2][0] = tube_C_sight_right[1][2] *
        tube_C_sight_right[0][1];
   tube_C_sight_right[2][1] = -tube_C_sight_right[1][2] *
        tube_C_sight_right[0][0];
   tube_C_sight_right[2][2] = mag;
    mat_copy (tube_C_sight_right, tube_C_sight_left);
   tube_C_sight_left[0][1] = -tube_C_sight_left[0][1];
   tube_C_sight_left[1][0] = -tube_C_sight_left[1][0];
   tube_C_sight_left[2][0] = -tube_C_sight_left[2][0];
```

The adat_temp_bias_coeff array is used to compute the temporal bias applied to the target location using the CSU missile_util_eval_poly, and called by the CSU missile_adat_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the target point, etc.
/*/
  if ((mptr->state == ADAT_GUIDE) || (mptr->state == ADAT_CLOSE))
  {
      if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state ==
                                                    ADAT_GUIDE))
    {
          bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
              adat_temp_bias_coeff, time);
      if (((tube / 2) * 2) == tube)
          missile_target_los_bias (mptr, sight_location,
              loc_sight_to_world, -bias, bias);
      else
          missile_target_los_bias (mptr, sight_location,
              loc_sight_to_world, bias, bias);
    }
    else
          missile_target_los (mptr, sight_location, loc_sight_to_world);
  }
  else if (mptr->state == ADAT_UNGUIDE)
      missile_target_unguided (mptr);
  else
      printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
```

See APPENDIX E for a complete source code listing.

3.36 Atgm_miss_char

The atgm_miss_char array consists of characteristics and parameters describing an ATGM missile system and its performance constraints. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

3.36.1 TOW_BURNOUT_TIME [for ATGM]

TOW_BURNOUT_TIME is a constant defining the time of powered flight for ATGM missile in ticks.

3.36.1.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_BURNOUT_TIME      tow_miss_char[ 0]
```

3.36.1.2 Usage

During real-time execution, this constant is not recomputed.

3.36.1.2.1 Algorithm

TOW_BURNOUT_TIME is used to control computation of the missile flyout speed and the cosines of the maximum allowed turn angles in each direction by a call to the CSC missile_atgm_fly.

```
/*/
 * Find the current missile speed and the cosines of the maximum
 *     allowed turn angles in each direction. The equations used are
 *     different before and after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
  {
     mptr->speed = mptr->init_speed +
       (speed_factor *
          missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                      tow_burn_speed_coeff, time));
       missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
     mptr->speed = mptr->init_speed +
       (speed_factor *
          missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                      tow_coast_speed_coeff, time));
       missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

See APPENDIX F for a complete source code listing.

3.36.2 TOW_RANGE_LIMIT_TIME [for ATGM]

TOW_RANGE_LIMIT_TIME is a constant defining the range limit time for the ATGM missile in ticks; at this point the wire is cut, but the missile is allowed to fly to the maximum flight time.

3.36.2.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_RANGE_LIMIT_TIME   tow_miss_char[ 1]
```

3.36.2.2 Usage

During real-time execution, this constant is not recomputed.

3.36.2.2.1 Algorithm

TOW_RANGE_LIMIT_TIME is used to control the wire cut at the maximum range flight time for an individual ATGM missile by a call to the CSU missile_atgm_fly.

```
/*/
 * If the missile has reached its maximum range (not the maximum distance
 * its allowed to fly), cut the wire.
/*/
   if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)
      tptr->wire_is_cut = TRUE;
```

See APPENDIX F for a complete source code listing.

3.36.3 TOW_MAX_FLIGHT_TIME [for ATGM]

TOW_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the ATGM missile assumed in ticks; cosine of the maximum turn angle is greater than 1.0 beyond this point.

3.36.3.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_MAX_FLIGHT_TIME   tow_miss_char[ 2]
```

3.36.3.2 Usage

During real-time execution, this constant is not recomputed.

3.36.3.2.1 Algorithm

TOW_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual ATGM missile by a call to the CSU missile_atgm_init.

```
    tptr->mptr.max_flight_time = TOW_MAX_FLIGHT_TIME;
```

See APPENDIX F for a complete source code listing.

3.36.4 ATGM_TURN_FACTOR

ATGM_TURN_FACTOR is a constant defining the ratio of the ATGM to TOW missile performance in turns; ATGM turn factor for wider turning capability with respect to TOW

3.36.4.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

```
#define ATGM_TURN_FACTOR       tow_miss_char[3]
```

3.36.4.2 Usage

During real-time execution, this constant is not recomputed.

3.36.4.2.1 Algorithm

ATGM_TURN_FACTOR is used to modify the tow burn turn and tow coast turn coefficients for each axis by a call to the CSU missile_atgm_init.

```
    /******************************************************************/
/* change turn polynomial coefficients so missile has larger    */
/* max turn angle.  Since Ph determines when a vehicle should be    */
/* impacted, turn rates should not effect missile effectiveness  */
    /******************************************************************/
for (i=0; i<tow_burn_turn_coeff.deg; i++)
    {
    tow_burn_turn_coeff.side_coeff[i]  *=  ATGM_TURN_FACTOR;
    tow_burn_turn_coeff.up_coeff[i]  *=  ATGM_TURN_FACTOR;
    tow_burn_turn_coeff.down_coeff[i]  *=  ATGM_TURN_FACTOR;
    }
for (i=0; i<tow_coast_turn_coeff.deg; i++)
    {
    tow_coast_turn_coeff.side_coeff[i]  *=  ATGM_TURN_FACTOR;
    tow_coast_turn_coeff.up_coeff[i]  *=  ATGM_TURN_FACTOR;
    tow_coast_turn_coeff.down_coeff[i]  *=  ATGM_TURN_FACTOR;
    }
```

See APPENDIX F for a complete source code listing.

3.37 Atgm_miss_poly_deg

The atgm_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the ATGM missile. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

3.37.1 TOW_BURN_SPEED_DEG [for ATGM]

TOW_BURN_SPEED_DEG is a constant defining the polynomial degree for the ATGM missile burn speed coefficient data array.

3.37.1.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by the CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define TOW_BURN_SPEED_DEG   tow_miss_poly_deg[0]
```

3.37.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW_BURN_SPEED_DEG is 4, especially, the declared size of the tow_burn_speed_coeff is 5.

3.37.1.2.1 Algorithm

TOW_BURN_SPEED_DEG is used to compute the ATGM missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_atgm_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
  mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  mat_copy (loc_sight_to_world, mptr->orientation);
  mptr->speed = launch_speed +
      (speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                                 tow_burn_speed_coeff, 0.0));
  mptr->init_speed = launch_speed;
```

TOW_BURN_SPEED_DEG is used to compute the ATGM missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_atgm_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
   {
        mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
             tow_burn_speed_coeff, time) + mptr->init_speed;
         missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
   }
   else
   {
        mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
             tow_coast_speed_coeff, time) + mptr->init_speed;
         missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
   }
```

See APPENDIX F for a complete source code listing.

3.37.2 TOW_COAST_SPEED_DEG [for ATGM]

TOW_COAST_SPEED_DEG is a constant defining the polynomial degree for ATGM missile coast speed coefficient data array

3.37.2.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by the CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_SPEED_DEG  tow_miss_poly_deg[1]
```

3.37.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW_COAST_SPEED_DEG is 4, especially, the declared size of the tow_burn_speed_coeff is 5.

3.37.2.2.1 Algorithm

TOW_COAST_SPEED_DEG is used to compute the ATGM missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_atgm_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
          tow_burn_speed_coeff, time) + mptr->init_speed;
      missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
          tow_coast_speed_coeff, time) + mptr->init_speed;
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

See APPENDIX F for a complete source code listing.

3.37.3 TOW_BURN_TURN_DEG [for ATGM]

TOW_BURN_TURN_DEG is a constant defining the polynomial degree for each ATGM missile burn turn coefficient data sub-array of the ATGM missile burn turn coefficient data array structure

3.37.3.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the constant data.

```
#define TOW_BURN_TURN_DEG    tow_miss_poly_deg[2]
```

3.37.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW_BURN_TURN_DEG is 1, especially, the declared size of the tow_burn_turn_coeff is 2. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
/*/
 * Coefficients for the cosine of max turn polynomials before motor burnout.
 * The structure _MAX_COS_COEFF_ is used to store the values for the turn
 * sideways, up, and down polynomials along with their order.
/*/

static MAX_COS_COEFF tow_burn_turn_coeff =
{
  1,            /* Order of the polynomials. */
  {
              /* Sidewards turn. */
    0.999976868652,  /* a_0 - cos(rad)/tick */
   -3.5933955e-7     /* a_1 - cos(rad)/tick**2 */
  },
  {
              /* Upwards turn. */
    0.999960667258,  /* a_0 - cos(rad)/tick */
   -3.1492328e-6     /* a_1 - cos(rad)/tick**2 */
  },
  {
              /* Downwards turn. */
    0.999978909989,  /* a_0 - cos(rad)/tick */
   -7.8194991e-9     /* a_1 - cos(rad)/tick**2 */
  }
};
```

3.24.3.2.1 Algorithm

TOW_BURN_TURN_DEG is hard coded by type definition of MAX_COS_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the ATGM missile during powered flight [burn] using the CSU missile_util_cos_coeff, and called by the CSU missile_atgm_fly. The CSU missile_util_cos_coeff uses the Newton-

Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
  {
       mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
            tow_burn_speed_coeff, time) + mptr->init_speed;
       missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
       mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
            tow_coast_speed_coeff, time) + mptr->init_speed;
       missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

See APPENDIX F for a complete source code listing.

3.37.4 TOW_COAST_TURN_DEG [for ATGM]

TOW_COAST_TURN_DEG is a constant defining the polynomial degree for each ATGM missile coast turn coefficient data sub-array of the ATGM missile coast turn coefficient data array structure

3.37.4.1 Initialization

The constant is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_TURN_DEG   tow_miss_poly_deg[3]
```

3.37.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value in each axis for TOW_COAST_TURN_DEG is 3, especially, the declared size of the tow_coast_turn_coeff is 4. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
/*/
 * Coefficients for the cosine of max turn polynomials after motor burnout.
/*/

static MAX_COS_COEFF tow_coast_turn_coeff =
{
  3,              /* Order of the polynomials. */
  {
                  /* Sidewards turn. */
    0.99995112518,  /* a_0 - cos(rad)/tick */
    8.96333e-7,     /* a_1 - cos(rad)/tick**2 */
   -5.995375e-9,    /* a_2 - cos(rad)/tick**3 */
    1.162225e-11    /* a_3 - cos(rad)/tick**4 */
  },
  {
                  /* Upwards turn. */
    0.9998498495,   /* a_0 - cos(rad)/tick */
    1.657779e-6,    /* a_1 - cos(rad)/tick**2 */
   -8.231861e-9,    /* a_2 - cos(rad)/tick**3 */
    1.381832e-11    /* a_3 - cos(rad)/tick**4 */
  },
  {
                  /* Downwards turn. */
    0.9999714014,   /* a_0 - cos(rad)/tick */
    3.382077e-7,    /* a_1 - cos(rad)/tick**2 */
   -1.601259e-9,    /* a_2 - cos(rad)/tick**3 */
    2.623014e-12    /* a_3 - cos(rad)/tick**4 */
  }
};
```

3.37.4.2.1 Algorithm

TOW_COAST_TURN_DEG is hard coded by type definition of MAX_COS_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the atgm missile during unpowered flight

[coast] using the CSU missile_util_cos_coeff, and called by the CSU missile_atgm_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
  {
       mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
            tow_burn_speed_coeff, time) + mptr->init_speed;
       missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
       mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
            tow_coast_speed_coeff, time) + mptr->init_speed;
       missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

See APPENDIX F for a complete source code listing.

3.38 Tow_burn_speed_coeff [for ATGM]

The tow_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the ATGM missile burn speed with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

3.38.1 Initialization

The tow_burn_speed_coeff array is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.38. - ATGM Missile Burn Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial before motor burnout initialized to
 * default values.
/*/

static REAL tow_burn_speed_coeff[5] =
{
   4.466666667,      /* a_0 - m/tick   ( 67.0 m/sec) */
   1.222103405,      /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
  -0.024532086,      /* a_2 - m/tick**3 (-82.7057910 m/sec**3) */
   0.0,
   0.0
};
```

The array has a maximum size of 5 elements.

3.38.2 Usage

During real-time execution, this array is not recomputed. TOW_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.38.2.1 Algorithm

Tow_burn_speed_coeff is used to compute the ATGM missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_atgm_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
   mptr->time = 0.0;
   vec_copy (launch_point, mptr->location);
   mat_copy (loc_sight_to_world, mptr->orientation);
   mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
         tow_burn_speed_coeff, 0.0) + launch_speed;
   mptr->init_speed = launch_speed;
```

Tow_burn_speed_coeff is used to compute the ATGM missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_atgm_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 *  Find the current missile speed and the cosines of the maximum allowed
 *  turn angles in each direction.  The equations used are different before and
 *  after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
   {
       mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
           tow_burn_speed_coeff, time) + mptr->init_speed;
       missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
   }
   else
   {
       mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
           tow_coast_speed_coeff, time) + mptr->init_speed;
       missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
   }
```

See APPENDIX F for a complete source code listing.

3.39 Tow_coast_speed_coeff [for ATGM]

The tow_coast_speed_coeff array consists of the coefficients for a polynomial equation defining the ATGM missile coast speed with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

3.39.1 Initialization

The tow_coast_speed_coeff array is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.38. - ATGM Missile Coast Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial after motor burnout initialized to
 * default values.
/*/

static REAL tow_coast_speed_coeff[5] =
{
   21.81905383,      /* a_0 - m/tick   (327.2858074 m/sec) */
   -9.5382019e-2,    /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
    2.4378222e-4,    /* a_2 - m/tick**3 ( 0.8227650 m/sec**3) */
   -2.6311111e-7,    /* a_3 - m/tick**4 ( -0.0133200 m/sec**4) */
    0.0
};
```

The array has a maximum size of 5 elements.

3.39.2 Usage

During real-time execution, this array is not recomputed. TOW_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.39.2.1 Algorithm

Tow_coast_speed_coeff is used to compute the ATGM missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_atgm_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
    {
        mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
            tow_burn_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
    }
    else
```

```
{
    mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
        tow_coast_speed_coeff, time) + mptr->init_speed;
    missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX F for a complete source code listing.

3.40 Tow_burn_turn_coeff [for ATGM]

The tow_burn_turn_coeff two-dimensional array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM missile maximum cosine of turn while powered with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code. A turn factor is used to scale the TOW coefficients for ATGM performance.

3.40.1 Initialization

The tow_burn_turn_coeff array is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.40. - ATGM Missile Burn Turn Coefficients Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the cosine of max turn polynomials before motor burnout.
 * The structure _MAX_COS_COEFF_ is used to store the values for the turn
 * sideways, up, and down polynomials along with their order.
/*/

static MAX_COS_COEFF tow_burn_turn_coeff =
{
   1,              /* Order of the polynomials. */
   {
                   /* Sidewards turn. */
     0.999976868652,  /* a_0 - cos(rad)/tick */
    -3.5933955e-7     /* a_1 - cos(rad)/tick**2 */
   },
   {
```

```
            /* Upwards turn. */
    0.999960667258,  /* a_0 - cos(rad)/tick */
   -3.1492328e-6     /* a_1 - cos(rad)/tick**2 */
  },
  {
            /* Downwards turn. */
    0.999978909989,  /* a_0 - cos(rad)/tick */
   -7.8194991e-9     /* a_1 - cos(rad)/tick**2 */
  }
};
```

The array has a maximum size of 3 by 2 elements.

Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

3.40.2 Usage

During real-time execution, this array is not recomputed. The size of the array in the type definition for MAX_COS_COEFF determines the number of elements of the array to be used in the polynomial evaluation.

3.40.2.1 Algorithm

The tow_burn_turn_coeff array is initialized and scaled for ATGM missile performance by a call to the CSU missile_atgm_init.

```
      /*******************************************************************/
  /* change turn polynomial coefficients so missile has larger    */
  /* max turn angle.  Since Ph determines when a vehicle should be    */
  /* impacted, turn rates should not effect missile effectiveness  */
      /*******************************************************************/
  for (i=0; i<tow_burn_turn_coeff.deg; i++)
      {
      tow_burn_turn_coeff.side_coeff[i] *=  ATGM_TURN_FACTOR;
      tow_burn_turn_coeff.up_coeff[i]  *=  ATGM_TURN_FACTOR;
      tow_burn_turn_coeff.down_coeff[i]  *=  ATGM_TURN_FACTOR;
      }
```

```
for (i=0; i<tow_coast_turn_coeff.deg; i++)
    {
    tow_coast_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
    tow_coast_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
    tow_coast_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
    }
```

Tow_burn_turn_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the ATGM missile during powered flight [burn] using the CSU missile_util_cos_coeff, and called by the CSU missile_atgm_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
  if (time < TOW_BURNOUT_TIME)
  {
      mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
          tow_burn_speed_coeff, time) + mptr->init_speed;
      missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
          tow_coast_speed_coeff, time) + mptr->init_speed;
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

See APPENDIX F for a complete source code listing.

3.41 Tow_coast _turn_coeff [for ATGM]

The tow_coast_turn_coeff two-dimensional array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM missile maximum cosine of turn while unpowered with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs

have the same name as in the TOW missile source code. A turn factor is used to scale the TOW coefficients for ATGM performance.

3.41.1 Initialization

The tow_coast_turn_coeff array is initialized during execution of the CSU missile_atgm_init, called by CSC weapons_init. Execution of the CSU missile_atgm_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.41. - ATGM Missile Coast Turn Coefficients Data Array for a summary of the array data.

The array has a maximum size of 3 by 4 elements.

The following is the default declaration.

```
/*/
 * Coefficients for the cosine of max turn polynomials after motor burnout.
/*/

static MAX_COS_COEFF tow_coast_turn_coeff =
{
   3,              /* Order of the polynomials. */
   {
                   /* Sidewards turn. */
      0.99995112518,  /* a_0 - cos(rad)/tick */
      8.96333e-7,     /* a_1 - cos(rad)/tick**2 */
     -5.995375e-9,    /* a_2 - cos(rad)/tick**3 */
      1.162225e-11    /* a_3 - cos(rad)/tick**4 */
   },
   {
                   /* Upwards turn. */
      0.9998498495,   /* a_0 - cos(rad)/tick */
      1.657779e-6,    /* a_1 - cos(rad)/tick**2 */
     -8.231861e-9,    /* a_2 - cos(rad)/tick**3 */
      1.381832e-11    /* a_3 - cos(rad)/tick**4 */
   },
   {
                   /* Downwards turn. */
      0.9999714014,   /* a_0 - cos(rad)/tick */
      3.382077e-7,    /* a_1 - cos(rad)/tick**2 */
     -1.601259e-9,    /* a_2 - cos(rad)/tick**3 */
      2.623014e-12    /* a_3 - cos(rad)/tick**4 */
   }
};
```

3.41.2 Usage

During real-time execution, this array is not recomputed. MAVERICK_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.41.2.1 Algorithm

The tow_coast_turn_coeff array is initialized and scaled for ATGM missile performance by a call to the CSU missile_atgm_init.

```
        /*****************************************************************/
  /* change turn polynomial coefficients so missile has larger     */
  /* max turn angle.  Since Ph determines when a vehicle should be    */
  /* impacted, turn rates should not effect missile effectiveness  */
        /*****************************************************************/
  for (i=0; i<tow_burn_turn_coeff.deg; i++)
      {
      tow_burn_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
      tow_burn_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
      tow_burn_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
      }
  for (i=0; i<tow_coast_turn_coeff.deg; i++)
      {
      tow_coast_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
      tow_coast_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
      tow_coast_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
      }
```

Tow_coast_turn_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the ATGM missile during unpowered flight [coast] using the CSU missile_util_cos_coeff, and called by the CSU missile_atgm_fly. The CSU missile_util_cos_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction.  The equations used are different before and
 * after motor burnout.
/*/
   if (time < TOW_BURNOUT_TIME)
  {
```

```
        mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
            tow_burn_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
            tow_coast_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
    }
```

See APPENDIX F for a complete source code listing.

3.42 Kem_miss_char

The kem_miss_char array consists of characteristics and parameters describing a KEM missile system and its performance constraints. The KEM missile source code was derived from the ADAT missile source code.

3.42.1 KEM_BURNOUT_TIME

KEM_BURNOUT_TIME is a constant defining the time of powered flight for kem missile in ticks.

3.42.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.42. - KEM Missile Characteristics Data Array for a summary of the constants data.

```
#define KEM_BURNOUT_TIME        kem_miss_char[0]
```

3.42.2 Usage

During real-time execution, this constant is not recomputed.

3.42.2.1 Algorithm

KEM_BURNOUT_TIME is used to control computation of the missile flyout speed by a call to the CSC missile_kem_fly.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
  if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
            kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
  }
```

See APPENDIX H for a complete source code listing.

3.42.2 KEM_MAX_FLIGHT_TIME

KEM_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the KEM missile in ticks.

3.42.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.42. - KEM Missile Characteristics Data Array for a summary of the constants data.

```
#define KEM_MAX_FLIGHT_TIME    kem_miss_char[1]
```

3.42.2 Usage

During real-time execution, this constant is not recomputed.

3.42.2.1 Algorithm

KEM_MAX_FLIGHT_TIME is used to initialize the maximum flight time for an individual KEM missile by a call to the CSU missile_kem_init.

```
for (i = 0; i < num_missiles; i++)
{
    kem_array[i].mptr.state = KEM_FREE;
    kem_array[i].mptr.max_flight_time = KEM_MAX_FLIGHT_TIME;
    kem_array[i].mptr.max_turn_directions = 1;
}
```

See APPENDIX H for a complete source code listing.

3.42.3 KEM_TO_MACH5_FACTOR

KEM_TO_MACH5_FACTOR is a constant defining the speed factor to raise missile performance from ADAT to KEM; just after burnout, the ADAT has a maximum velocity of 230 m/sec, while the KEM has a maximum velocity of 1524 m/sec.

3.42.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.42. - KEM Missile Characteristics Data Array for a summary of the constants data.

```
#define KEM_TO_MACH5_FACTOR    kem_miss_char[2]
```

3.42.2 Usage

During real-time execution, this constant is not recomputed.

3.42.2.1 Algorithm

KEM_TO_MACH5_FACTOR is used to scale the burn speed coefficients when the launch speed is computed by a call to the CSU missile_kem_fire.

```
/*
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
*/
  mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  mat_copy (loc_sight_to_world, mptr->orientation);

  mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
        kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) +
      launch_speed;
  mptr->init_speed = launch_speed;

  if (kptr->target_vehicle_id.vehicle == vehicleIrrelevant)
     comm_target_type = targetUnknown;
  else
     comm_target_type = targetIsVehicle;
```

KEM_TO_MACH5_FACTOR is used to scale the burn speed and coast speed coefficients when the missile flyout speed is computed by a call to the CSU missile_kem_fly.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
*/
  if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
  }
```

```
else
{
    mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
    mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG,kem_coast_turn_coeff, time);
}
```

See APPENDIX H for a complete source code listing.

3.43 Kem_miss_poly_deg

The kem_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the KEM missile. The KEM missile source code was derived from the ADAT missile source code.

3.43.1 KEM_BURN_SPEED_DEG

KEM_BURN_SPEED_DEG is a constant defining the polynomial degree for the KEM missile burn speed coefficient data array. KEM_BURN_SPEED_DEG is the first element of the kem_miss_poly_deg array.

3.43.1.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define KEM_BURN_SPEED_DEG    kem_miss_poly_deg[0]
```

3.43.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM_BURN_SPEED_DEG is 9, especially, the declared size of the kem_burn_speed_coeff array is 10.

3.43.1.2.1 Algorithm

KEM_BURN_SPEED_DEG is used to compute the KEM missile speed at launch using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
 */
  mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  mat_copy (loc_sight_to_world, mptr->orientation);

  mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
        kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) +
      launch_speed;
  mptr->init_speed = launch_speed;

  if (kptr->target_vehicle_id.vehicle == vehicleIrrelevant)
     comm_target_type = targetUnknown;
  else
     comm_target_type = targetIsVehicle;
```

KEM_BURN_SPEED_DEG is used to compute the KEM missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
  if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
```

```
        mptr->cos_max_turn[0] = missile_util_eval_poly (
                KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
                kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
                mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (
                KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
    }
```

See APPENDIX H for a complete source code listing.

3.43.2 KEM_COAST_SPEED_DEG

KEM_COAST_SPEED_DEG is a constant defining the polynomial degree for the kem missile coast speed coefficient data array. KEM_COAST_SPEED_DEG is the second element of the kem_miss_poly_deg array.

3.43.2.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define KEM_COAST_SPEED_DEG    kem_miss_poly_deg[1]
```

3.43.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM_COAST_SPEED_DEG is 9, especially, the declared size of the kem_burn_speed_coeff array is 10.

3.43.2.2.1 Algorithm

KEM_COAST_SPEED_DEG is used to compute the KEM missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
  if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
            kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
  }
```

See APPENDIX H for a complete source code listing.

3.43.3 KEM_BURN_TURN_DEG

KEM_BURN_TURN_DEG is a constant defining the polynomial degree for the cosine of the KEM missile maximum allowed turn angle, burn turn coefficient data array. KEM_BURN_TURN_DEG is the third element of the kem_miss_poly_deg array.

3.43.3.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define  KEM_BURN_TURN_DEG     kem_miss_poly_deg[2]
```

3.43.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM_BURN_TURN_DEG is 9, especially, the declared size of the kem_burn_speed_coeff array is 10.

3.43.3.2.1 Algorithm

KEM_BURN_TURN_DEG is used to compute the cosine of the maximum allowed turn angle for the KEM missile during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
  if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
            kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
  }
```

See APPENDIX H for a complete source code listing.

3.43.4 KEM_COAST_TURN_DEG

KEM_COAST_TURN_DEG is a constant defining the polynomial degree for the cosine of the KEM missile maximum allowed turn angle, coast turn

coefficient data array. KEM_COAST_TURN_DEG is the fourth element of the kem_miss_poly_deg array.

3.43.4.1 Initialization

The constant is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define KEM_COAST_TURN_DEG   kem_miss_poly_deg[3]
```

3.43.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM_COAST_TURN_DEG is 9, especially, the declared size of the kem_burn_speed_coeff array is 10.

3.43.4.2.1 Algorithm

KEM_COAST_TURN_DEG is used to compute the cosine of the maximum allowed turn angle for the KEM missile during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
   if (time < KEM_BURNOUT_TIME)
   {
       mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
             kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
             mptr->init_speed;
       mptr->cos_max_turn[0] = missile_util_eval_poly (
             KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
   }
   else
   {
```

```
    mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
        kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
        mptr->init_speed;
    mptr->cos_max_turn[0] = missile_util_eval_poly (
        KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
}
```

See APPENDIX H for a complete source code listing.

3.44 Kem_burn_speed_coeff

The kem_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the KEM missile burn speed with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

3.44.1 Initialization

The kem_burn_speed_coeff array is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.44. - KEM Missile Burn Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial before motor burnout initialized
 * to default values.
/*/
```

```
static REAL kem_burn_speed_coeff[10] =
{
  2.296,          /* a_0 - m/tick */
  0.72990856,     /* a_1 - m/tick**2 */
  0.013310932,    /* a_2 - m/tick**3 */
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 10 elements.

3.44.2 Usage

During real-time execution, this array is not recomputed. KEM_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.44.2.1 Algorithm

The kem_burn_speed_coeff array is used to compute the initial speed at launch of the KEM missile using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fire. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
      kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) +
   launch_speed;
```

The kem_burn_speed_coeff array is used to compute the KEM missile speed during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
 if (time < KEM_BURNOUT_TIME)
 {
     mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
           kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
           mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
           KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
 }
 else
 {
     mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
           kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
           mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
           KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
 }
```

See APPENDIX H for a complete source code listing.

3.45 Kem_coast_speed_coeff

The kem_coast_speed_coeff array consists of the coefficients for a polynomial equation defining the KEM missile coast speed with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

3.45.1 Initialization

The kem_coast_speed_coeff array is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.45. - KEM Missile Coast Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial after motor burnout.
/*/

static REAL kem_coast_speed_coeff[10] =
{
 105.52162,        /* a_0 - m/tick */
  -1.0157285,      /* a_1 - m/tick**2 */
   5.6124330e-3,   /* a_2 - m/tick**3 */
  -1.6262608e-5,   /* a_3 - m/tick**4 */
   1.8991982e-8,   /* a_4 - m/tick**5 */
   0.0,
   0.0,
   0.0,
   0.0,
   0.0
};
```

The array has a maximum size of 10 elements.

3.45.2 Usage

During real-time execution, this array is not recomputed. KEM_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.45.2.1 Algorithm

The kem_coast_speed_coeff array is used to compute the KEM missile speed during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
 */
   if (time < KEM_BURNOUT_TIME)
   {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
```

```
            mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
            kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
    }
```

See APPENDIX H for a complete source code listing.

3.46 Kem_burn_turn_coeff

The kem_burn_turn_coeff array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of the turn angle while in powered flight with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

3.46.1 Initialization

The kem_burn_turn_coeff array is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.46. - KEM Missile Burn Turn Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the cosine of max turn polynomial before motor burnout.
/*/

static REAL kem_burn_turn_coeff[10] =
{
   0.999993,          /* a_0 - cos(rad)/tick */
  -6.2386917e-7,      /* a_1 - cos(rad)/tick**2 */
   1.6146426e-7,      /* a_2 - cos(rad)/tick**3 */
  -9.720142e-7,       /* a_3 - cos(rad)/tick**4 */
   0.0,
```

```
  0.0,
  0.0,
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 10 elements.

3.46.2 Usage

During real-time execution, this array is not recomputed. KEM_BURN_TURN_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.46.2.1 Algorithm

The kem_burn_turn_coeff array is used to compute the cosine of the maximum allowed turn angle for the KEM missile during powered flight [burn] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
*/
   if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
  }
```

```
else
{
    mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
    mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
}
```

See APPENDIX H for a complete source code listing.

3.47 Kem_coast_turn_coeff

The kem_coast_turn_coeff array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of the turn angle while in unpowered flight with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

3.47.1 Initialization

The kem_coast_turn_coeff array is initialized during execution of the CSU missile_kem_init, called by CSC weapons_init. Execution of the CSU missile_kem_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.47. - KEM Missile Coast Turn Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the cosine of max turn polynomial after motor burnout.
/*/

static REAL kem_coast_turn_coeff[10] =
{
    0.99753111,       /* a_0 - cos(rad)/tick */
    5.5817986e-5,     /* a_1 - cos(rad)/tick**2 */
   -5.1276276e-7,     /* a_2 - cos(rad)/tick**3 */
    2.2388593e-9,     /* a_3 - cos(rad)/tick**4 */
   -5.1964622e-12,    /* a_4 - cos(rad)/tick**5 */
    4.5499104e-15,    /* a_5 - cos(rad)/tick**6 */
```

```
   0.0,
   0.0,
   0.0,
   0.0
};
```

The array has a maximum size of 10 elements.

3.47.2 Usage

During real-time execution, this array is not recomputed. MAVERICK_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.47.2.1 Algorithm

The kem_coast_turn_coeff array is used to compute the cosine of the maximum allowed turn angle for the KEM missile during unpowered flight [coast] using the CSU missile_util_eval_poly, and called by the CSU missile_kem_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*
 * Find the current missile speed and the cosines of the maximum allowed
 * turn angles in each direction. The equations used are different before
 * and after motor burnout.
*/
   if (time < KEM_BURNOUT_TIME)
  {
      mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
  }
  else
  {
      mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
            kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
            mptr->init_speed;
      mptr->cos_max_turn[0] = missile_util_eval_poly (
            KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
  }
```

See APPENDIX H for a complete source code listing.

3.48 Nlos_miss_char

The nlos_miss_char array consists of characteristics and parameters describing an NLOS missile system and its performance constraints.

3.48.1 NLOS_LOCK_THRESHOLD

NLOS_LOCK_THRESHOLD is a constant defining the threshold lock for the NLOS missile

3.48.1.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define NLOS_LOCK_THRESHOLD          nlos_miss_char[ 0]
```

3.48.1.2 Usage

During real-time execution, this constant is not recomputed.

3.48.1.2.1 Algorithm

NLOS_LOCK_THRESHOLD is used to compute the vector to the preferred vehicle by a call to the CSU near_get_preferred_veh_near_vector in a call to the CSC missile_nlos_fly.

```
/*/
 * choose the correct targeting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
    printf("extra_waypoint: %f %f %f\n",
        mptr->location[0],
        mptr->location[1],
        mptr->location[2]);

    if (time < NLOS_VERTICAL_FLIGHT_TIME)
```

```
    missile_nlos_fly_to_point(mptr, peak_target);
else if (time < NLOS_DECLINE_FLIGHT_TIME)
    missile_nlos_fly_to_point(mptr, decline_target);
else if (time < NLOS_LEVEL_FLIGHT_TIME)
    {
    level_target[Z] = mptr->location[Z];
    missile_nlos_fly_to_point(mptr, level_target);
    }
else
    {
    switch (target_scheme)
      {
       case NLOS_FLY_TO_POINT_IN_SPACE:
          missile_nlos_fly_to_point(mptr, nlos_target_loc);
          break;

       case NLOS_FLY_TO_POINT_RELATIVE:
          missile_target_nlos(mptr, nlos_target_loc);
          break;

       case NLOS_FLY_TO_TARGET:
          target = near_get_preferred_veh_near_vector (
                              &nlos_target_id,
                              RVA_ALL_VEH,
                              mptr->location,
                              mptr->orientation[1],
                              NLOS_LOCK_THRESHOLD,
                              &nlos_req_id);
          if (target != NULL)
            {
             timed_printf("miss_nlos: target locked on\n");
       missile_target_pursuit (mptr, target);
            }
          else
            {
             missile_target_unguided(mptr);
            }
          break;

       default:
          printf("missile_nlos_fly: bad target_scheme\n");
          break;
      }
    }
```

See APPENDIX J for a complete source code listing.

3.48.2 NLOS_MAX_TURN_ANGLE

NLOS_MAX_TURN_ANGLE is a constant defining the maximum turn angle for the NLOS missile.

3.48.2.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define  NLOS_MAX_TURN_ANGLE            nlos_miss_char[ 1]
```

3.48.2.2 Usage

During real-time execution, this constant is not recomputed.

3.48.2.2.1 Algorithm

NLOS_MAX_TURN_ANGLE is used to compute the cosine of the maximum turn angle for the NLOS missile by a call to the CSU missile_nlos_init.

```
mptr->cos_max_turn[0] = cos (NLOS_MAX_TURN_ANGLE);
```

See APPENDIX J for a complete source code listing.

3.48.3 NLOS_VERTICAL_FLIGHT_TIME

NLOS_VERTICAL_FLIGHT_TIME is a constant defining the flight time in the vertical mode for the NLOS missile.

3.48.3.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define NLOS_VERTICAL_FLIGHT_TIME  nlos_miss_char[ 2]
```

3.48.3.2 Usage

During real-time execution, this constant is not recomputed.

3.48.3.2.1 Algorithm

NLOS_VERTICAL_FLIGHT_TIME is used control the flight path of the NLOS missile by a call to the CSC missile_nlos_fly.

```
/*/
 * choose the correct targeting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
  printf("extra_waypoint: %f %f %f\n",
      mptr->location[0],
      mptr->location[1],
      mptr->location[2]);

  if (time < NLOS_VERTICAL_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, peak_target);
  else if (time < NLOS_DECLINE_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, decline_target);
  else if (time < NLOS_LEVEL_FLIGHT_TIME)
      {
      level_target[Z] = mptr->location[Z];
      missile_nlos_fly_to_point(mptr, level_target);
      }
  else
      {
      switch (target_scheme)
        {
         case NLOS_FLY_TO_POINT_IN_SPACE:
            missile_nlos_fly_to_point(mptr, nlos_target_loc);
            break;

         case NLOS_FLY_TO_POINT_RELATIVE:
            missile_target_nlos(mptr, nlos_target_loc);
            break;
```

```
        case NLOS_FLY_TO_TARGET:
            target = near_get_preferred_veh_near_vector (
                                &nlos_target_id,
                                RVA_ALL_VEH,
                                mptr->location,
                                mptr->orientation[1],
                                NLOS_LOCK_THRESHOLD,
                                &nlos_req_id);
            if (target != NULL)
              {
               timed_printf("miss_nlos: target locked on\n");
        missile_target_pursuit (mptr, target);
              }
            else
              {
               missile_target_unguided(mptr);
              }
            break;

        default:
            printf("missile_nlos_fly: bad target_scheme\n");
            break;
        }
    }
```

See APPENDIX J for a complete source code listing.

3.48.4 NLOS_DECLINE_FLIGHT_TIME

NLOS_DECLINE_FLIGHT_TIME is a constant defining the flight time in the decline mode for the NLOS missile.

3.48.4.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define NLOS_DECLINE_FLIGHT_TIME     nlos_miss_char[ 3]
```

3.48.4.2 Usage

During real-time execution, this constant is not recomputed.

3.48.4.2.1 Algorithm

NLOS_DECLINE_FLIGHT_TIME is used control the flight path of the NLOS missile by a call to the CSC missile_nlos_fly.

```
/*/
 * choose the correct targeting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
   printf("extra_waypoint: %f %f %f\n",
       mptr->location[0],
       mptr->location[1],
       mptr->location[2]);

   if (time < NLOS_VERTICAL_FLIGHT_TIME)
       missile_nlos_fly_to_point(mptr, peak_target);
   else if (time < NLOS_DECLINE_FLIGHT_TIME)
       missile_nlos_fly_to_point(mptr, decline_target);
   else if (time < NLOS_LEVEL_FLIGHT_TIME)
       {
       level_target[Z] = mptr->location[Z];
       missile_nlos_fly_to_point(mptr, level_target);
       }
   else
       {
       switch (target_scheme)
         {
          case NLOS_FLY_TO_POINT_IN_SPACE:
             missile_nlos_fly_to_point(mptr, nlos_target_loc);
             break;

          case NLOS_FLY_TO_POINT_RELATIVE:
             missile_target_nlos(mptr, nlos_target_loc);
             break;
```

```
        case NLOS_FLY_TO_TARGET:
            target = near_get_preferred_veh_near_vector (
                                &nlos_target_id,
                                RVA_ALL_VEH,
                                mptr->location,
                                mptr->orientation[1],
                                NLOS_LOCK_THRESHOLD,
                                &nlos_req_id);
            if (target != NULL)
              {
                timed_printf("miss_nlos: target locked on\n");
        missile_target_pursuit (mptr, target);
              }
            else
              {
                missile_target_unguided(mptr);
              }
            break;

        default:
            printf("missile_nlos_fly: bad target_scheme\n");
            break;
        }
    }
```

See APPENDIX J for a complete source code listing.

3.48.5 NLOS_LEVEL_FLIGHT_TIME

NLOS_LEVEL_FLIGHT_TIME is a constant defining the flight time in the level mode for the NLOS missile.

3.48.5.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define NLOS_LEVEL_FLIGHT_TIME            nlos_miss_char[ 4]
```

3.48.5.2 Usage

During real-time execution, this constant is not recomputed.

3.48.5.2.1 Algorithm

NLOS_LEVEL_FLIGHT_TIME is used control the flight path of the NLOS missile by a call to the CSC missile_nlos_fly.

```
/*/
 * choose the correct targeting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
   printf("extra_waypoint: %f %f %f\n",
       mptr->location[0],
       mptr->location[1],
       mptr->location[2]);

   if (time < NLOS_VERTICAL_FLIGHT_TIME)
       missile_nlos_fly_to_point(mptr, peak_target);
   else if (time < NLOS_DECLINE_FLIGHT_TIME)
       missile_nlos_fly_to_point(mptr, decline_target);
   else if (time < NLOS_LEVEL_FLIGHT_TIME)
       {
       level_target[Z] = mptr->location[Z];
       missile_nlos_fly_to_point(mptr, level_target);
       }
  else
       {
       switch (target_scheme)
         {
          case NLOS_FLY_TO_POINT_IN_SPACE:
              missile_nlos_fly_to_point(mptr, nlos_target_loc);
              break;

          case NLOS_FLY_TO_POINT_RELATIVE:
              missile_target_nlos(mptr, nlos_target_loc);
              break;
```

```
        case NLOS_FLY_TO_TARGET:
            target = near_get_preferred_veh_near_vector (
                                &nlos_target_id,
                                RVA_ALL_VEH,
                                mptr->location,
                                mptr->orientation[1],
                                NLOS_LOCK_THRESHOLD,
                                &nlos_req_id);
            if (target != NULL)
              {
               timed_printf("miss_nlos: target locked on\n");
        missile_target_pursuit (mptr, target);
              }
            else
              {
               missile_target_unguided(mptr);
              }
            break;

        default:
            printf("missile_nlos_fly: bad target_scheme\n");
            break;
        }
    }
```

See APPENDIX J for a complete source code listing.

3.48.6 NLOS_ARM_TIME

NLOS_ARM_TIME is a constant defining the nlos missile arm time delay before firing in ticks.

3.48.6.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define  NLOS_ARM_TIME                    nlos_miss_char[ 5]
```

3.48.6.2 Usage

During real-time execution, this constant is not recomputed.

3.48.6.2.1 Algorithm

NLOS_ARM_TIME is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.7 NLOS_BURNOUT_TIME

NLOS_BURNOUT_TIME is a constant defining the time of powered flight for the nlos missile in ticks.

3.48.7.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define  NLOS_BURNOUT_TIME            nlos_miss_char[ 6]
```

3.48.7.2 Usage

During real-time execution, this constant is not recomputed.

3.48.7.2.1 Algorithm

NLOS_BURNOUT_TIME is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.8 NLOS_MAX_FLIGHT_TIME

NLOS_MAX_FLIGHT_TIME is a constant defining the maximum flight time for the nlos missile assumed in ticks.

3.48.8.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define NLOS_MAX_FLIGHT_TIME          nlos_miss_char[ 7]
```

3.48.8.2 Usage

During real-time execution, this constant is not recomputed.

3.48.8.2.1 Algorithm

NLOS_MAX_FLIGHT_TIME is used to initialize the maximum flight time for the NLOS missile in the CSU missile_nlos_init.

```
    mptr->max_flight_time = NLOS_MAX_FLIGHT_TIME;
```

See APPENDIX J for a complete source code listing.

3.48.9 SPEED_0

SPEED_0 is a constant defining the reference speed for the NLOS missile.

3.48.9.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define SPEED_0                       nlos_miss_char[ 8]
```

3.48.9.2 Usage

During real-time execution, this constant is not recomputed.

3.48.9.2.1 Algorithm

SPEED_0 is used to initialize the speed for the NLOS missile in calls to the CSU missile_nlos_init and the CSU missile_nlos_fire.

```
mptr->speed = SPEED_0;
```

See APPENDIX J for a complete source code listing.

3.48.10 SPEED_1

SPEED_1 is a constant defining the second speed profile of the NLOS missile during flight.

3.48.10.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define SPEED_1                         nlos_miss_char[ 9]
```

3.48.10.2 Usage

During real-time execution, this constant is not recomputed.

3.48.10.2.1 Algorithm

SPEED_1 is used to initialize the NLOS flight speed during the second phase of the flyout after time from launch exceeds 800 ticks.

```
/*/
 * Set and _time_. This is created mostly for increased readablity.
/*/
   time = mptr->time;

   if (time > 800.0)
        mptr->speed = SPEED_1;
```

See APPENDIX J for a complete source code listing.

3.48.11 THETA_0

THETA_0 is a constant defining the reference maximum turn angle which is scaled for speed.

3.48.11.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
/*#define THETA_0    0.046542113 */    /*0.013962634*/
#define THETA_0                       nlos_miss_char[10]
```

3.48.11.2 Usage

During real-time execution, this constant is not recomputed.

3.48.11.2.1 Algorithm

THETA_0 is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.12 SIN_UNGUIDE

SIN_UNGUIDE is a constant defining the sine of level flight [4.0 degrees pitch] for an unguided nlos missile.

3.48.12.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_UNGUIDE                   nlos_miss_char[11]
```

3.48.12.2 Usage

During real-time execution, this constant is not recomputed.

3.48.12.2.1 Algorithm

SIN_UNGUIDE is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.13 COS_UNGUIDE

COS_UNGUIDE is a constant defining the cosine of level flight [4.0 degrees pitch] for an unguided nlos missile.

3.48.13.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_UNGUIDE                nlos_miss_char[12]
```

3.48.13.2 Usage

During real-time execution, this constant is not recomputed.

3.48.13.2.1 Algorithm

COS_UNGUIDE is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.14 SIN_CLIMB

SIN_CLIMB is a constant defining the sine of the delta pitch angle [3.5 degrees] for a climbing nlos missile.

3.48.14.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_CLIMB                    nlos_miss_char[13]
```

3.48.14.2 Usage

During real-time execution, this constant is not recomputed.

3.48.14.2.1 Algorithm

SIN_CLIMB is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.15 COS_CLIMB

COS_CLIMB is a constant defining the cosine of the delta pitch angle [3.5 degrees] for a climbing nlos missile.

3.48.15.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_CLIMB                    nlos_miss_char[14]
```

3.48.15.2 Usage

During real-time execution, this constant is not recomputed.

3.48.15.2.1 Algorithm

COS_CLIMB is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.16 SIN_LOCK

SIN_LOCK is a constant defining the sine of the lock cone angle [9.0 degrees] for a locked-on nlos missile.

3.48.16.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define SIN_LOCK                    nlos_miss_char[15]
```

3.48.16.2 Usage

During real-time execution, this constant is not recomputed.

3.48.16.2.1 Algorithm

SIN_LOCK is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.17 COS_LOCK

COS_LOCK is a constant defining the cosine of the lock cone angle [9.0 degrees] for a locked-on nlos missile.

3.48.17.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOCK                    nlos_miss_char[16]
```

3.48.17.2 Usage

During real-time execution, this constant is not recomputed.

3.48.17.2.1 Algorithm

COS_LOCK is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.18 COS_TERM

COS_TERM is a constant defining the cosine of the terminal angle [0.0 degrees] for a locked-on nlos missile.

3.48.18.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_TERM                    nlos_miss_char[17]
```

3.48.18.2 Usage

During real-time execution, this constant is not recomputed.

3.48.18.2.1 Algorithm

COS_TERM is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.48.19 COS_LOSE

COS_LOSE is a constant defining the cosine of the angle [20.0 degrees] for a loss-of-lock-on nlos missile.

3.48.19.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOSE                    nlos_miss_char[18]
```

3.48.19.2 Usage

During real-time execution, this constant is not recomputed.

3.48.19.2.1 Algorithm

COS_LOSE is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.49 Nlos_miss_poly_deg

The nlos_miss_poly_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, and the coast speed for the NLOS missile.

3.49.1 NLOS_BURN_SPEED_DEG

NLOS_BURN_SPEED_DEG is a constant defining the polynomial degree for the NLOS missile burn speed coefficient data array

3.49.1.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.49. - NLOS Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define NLOS_BURN_SPEED_DEG     nlos_miss_poly_deg[0]
```

3.49.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for NLOS_BURN_SPEED_DEG is 4, especially, the declared size of the nlos_burn_speed_coeff array is 5.

3.49.1.2.1 Algorithm

NLOS_BURN_SPEED_DEG is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.49.2 NLOS_COAST_SPEED_DEG

NLOS_COAST_SPEED_DEG is a constant defining the polynomial degree for the NLOS missile coast speed coefficient data array.

3.49.2.1 Initialization

The constant is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.49. - NLOS Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define NLOS_COAST_SPEED_DEG   nlos_miss_poly_deg[1]
```

3.49.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for NLOS_COAST_SPEED_DEG is 4, especially, the declared size of the nlos_coast_speed_coeff array is 5.

3.49.2.2.1 Algorithm

NLOS_COAST_SPEED_DEG is not used in the current calculations.

See APPENDIX J for a complete source code listing.

3.50 Nlos_burn_speed_coeff

The nlos_burn_speed_coeff array consists of the coefficients for a polynomial equation defining the NLOS missile burn speed with respect to time in the form using the Newton-Raphson method.

3.50.1 Initialization

The nlos_burn_speed_coeff array is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.50. - NLOS Missile Burn Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial before motor burnout.
/*/

static REAL nlos_burn_speed_coeff[5] =
{
  0.03333333,      /* a_0 - m/tick   ( 67.0 m/sec)      */
  1.25777777,      /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 5 elements.

3.50.2 Usage

During real-time execution, this array is not recomputed. NLOS_BURN_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.50.2.1 Algorithm

The nlos_burn_speed_coeff array is not used in the current calculations for NLOS missile burn speed. The NLOS missile speed profile is constant in phase one and phase two, using a time as the delimiter.

See APPENDIX J for a complete source code listing.

3.51 Nlos_coast_speed_coeff

The nlos_coast_speed_coeff array consists of the coefficients for a polynomial equation defining the NLOS missile coast speed with respect to time in the form using the Newton-Raphson method.

3.51.1 Initialization

The nlos_coast_speed_coeff array is initialized during execution of the CSU missile_nlos_init, called by CSC weapons_init. Execution of the CSU missile_nlos_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.51. - NLOS Missile Coast Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
 * Coefficients for the speed polynomial after motor burnout.
/*/

static REAL nlos_coast_speed_coeff[5] =
{
   30.46972849,     /* a_0 - m/tick    (327.2858074 m/sec)    */
   -9.7721160e-2,   /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
    1.2433925e-4,   /* a_2 - m/tick**3 ( 0.8227650 m/sec**3)  */
   -5.4061501e-8,   /* a_3 - m/tick**4 ( -0.0133200 m/sec**4) */
    0.0
};
```

The array has a maximum size of 5 elements.

3.51.2 Usage

During real-time execution, this array is not recomputed. NLOS_COAST_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.51.2.1 Algorithm

The nlos_coast_speed_coeff array is not used in the current calculations for NLOS missile coast speed. The NLOS missile speed profile is constant in phase one and phase two, using a time as the delimiter.

See APPENDIX J for a complete source code listing.

3.52 Hydra_rkt_char

The hydra_rkt_char array consists of characteristics and parameters describing a rocket launcher system and its performance constraints.

3.52.1 HYDRA_LAUNCHER_POS_X

HYDRA_LAUNCHER_POS_X is a constant defining the hydra launcher position in the x-axis.

3.52.1.1 Initialization

The constant is initialized during execution of the CSU hydra_init, called by CSC weapons_init. Execution of the CSU hydra_init is normally done only

once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HYDRA_LAUNCHER_POS_X        hydra_rkt_char[0]
```

3.52.1.2 Usage

During real-time execution, this constant is not recomputed.

3.52.1.2.1 Algorithm

HYDRA_LAUNCHER_POS_X is used to initialize the left and right launcher position, rotational elements, and offset positions in the x-axis in a call to the CSU hydra_init.

```
,    left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
     right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
     articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
     articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;

  if(!rotate_init_element( &articulation_element, hull(),
              1.0, 0.0, 0.0, 0.0,

ARTICULATION_MIN,ARTICULATION_MAX,/*TWO_*/PI,/*rate*/
              0.0, HYDRA_LAUNCHER_POS_Y,
             HYDRA_LAUNCHER_POS_Z ))
  {
     printf( "Rotate_Init_Element: articulation_element FAILED\n" );
  }

  rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
              -TWO_PI, TWO_PI, TWO_PI, /*rate*/
              -HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
  rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
              -TWO_PI, TWO_PI, TWO_PI, /*rate*/
              HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
  missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
   missile_hydra_set_pylon_position_offsets(
                          HYDRA_LAUNCHER_POS_X,
                          HYDRA_LAUNCHER_POS_Y,
                          HYDRA_LAUNCHER_POS_Z );
```

See APPENDIX N for a complete source code listing.

3.52.2 HYDRA_LAUNCHER_POS_Y

HYDRA_LAUNCHER_POS_Y is a constant defining the hydra launcher position in the y-axis.

3.52.2.1 Initialization

The constant is initialized during execution of the CSU hydra_init, called by CSC weapons_init. Execution of the CSU hydra_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HYDRA_LAUNCHER_POS_Y        hydra_rkt_char[1]
```

3.52.2.2 Usage

During real-time execution, this constant is not recomputed.

3.52.2.2.1 Algorithm

HYDRA_LAUNCHER_POS_Y is used to initialize the left and right launcher position, rotational elements, and offset positions in the y-axis in a call to the CSU hydra_init.

```
        left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
        right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
        articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
        articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;

    if(!rotate_init_element( &articulation_element, hull(),
                1.0, 0.0, 0.0, 0.0,

ARTICULATION_MIN,ARTICULATION_MAX,/*TWO_*/PI,/*rate*/
                0.0, HYDRA_LAUNCHER_POS_Y,
              HYDRA_LAUNCHER_POS_Z ))
    {
        printf( "Rotate_Init_Element: articulation_element FAILED\n" );
    }
```

```
rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
            -TWO_PI, TWO_PI, TWO_PI, /*rate*/
            -HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
            -TWO_PI, TWO_PI, TWO_PI, /*rate*/
            HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
 missile_hydra_set_pylon_position_offsets(
                        HYDRA_LAUNCHER_POS_X,
                        HYDRA_LAUNCHER_POS_Y,
                        HYDRA_LAUNCHER_POS_Z );
```

See APPENDIX N for a complete source code listing.

3.52.3 HYDRA_LAUNCHER_POS_Z

HYDRA_LAUNCHER_POS_Z is a constant defining the hydra launcher position in the z-axis.

3.52.3.1 Initialization

The constant is initialized during execution of the CSU hydra_init, called by CSC weapons_init. Execution of the CSU hydra_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HYDRA_LAUNCHER_POS_Z        hydra_rkt_char[2]
```

3.52.3.2 Usage

During real-time execution, this constant is not recomputed.

3.52.3.2.1 Algorithm

HYDRA_LAUNCHER_POS_Z is used to initialize the left and right launcher position, rotational elements, and offset positions in the z-axis in a call to the CSU hydra_init.

```
left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;
```

```
    if(!rotate_init_element( &articulation_element, hull(),
                1.0, 0.0, 0.0, 0.0,

ARTICULATION_MIN,ARTICULATION_MAX,/*TWO_*/PI,/*rate*/
                0.0, HYDRA_LAUNCHER_POS_Y,
              HYDRA_LAUNCHER_POS_Z ))
  {
      printf( "Rotate_Init_Element: articulation_element FAILED\n" );
  }

   rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
                -TWO_PI, TWO_PI, TWO_PI, /*rate*/
                -HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
   rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
                -TWO_PI, TWO_PI, TWO_PI, /*rate*/
                HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
   missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
    missile_hydra_set_pylon_position_offsets(
                           HYDRA_LAUNCHER_POS_X,
                           HYDRA_LAUNCHER_POS_Y,
                           HYDRA_LAUNCHER_POS_Z );
```

See APPENDIX N for a complete source code listing.

3.52.4 SOVIET_ARTICULATION

SOVIET_ARTICULATION is a constant defining the angle of Soviet articulation in mils.

3.52.4.1 Initialization

The constant is initialized during execution of the CSU hydra_init, called by CSC weapons_init. Execution of the CSU hydra_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define SOVIET_ARTICULATION      ( mil_to_rad(hydra_rkt_char[3]))
```

3.52.4.2 Usage

During real-time execution, this constant is not recomputed.

3.52.4.2.1 Algorithm

SOVIET_ARTICULATION is not used in the current calculations.

See APPENDIX N for a complete source code listing.

3.52.5 HULL_NEG_5_PITCH

HULL_NEG_5_PITCH is a constant defining the degrees of hull negative pitch.

3.52.5.1 Initialization

The constant is initialized during execution of the CSU hydra_init, called by CSC weapons_init. Execution of the CSU hydra_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HULL_NEG_5_PITCH        ( deg_to_rad(hydra_rkt_char[4]))
```

3.52.5.2 Usage

During real-time execution, this constant is not recomputed.

3.52.5.2.1 Algorithm

HULL_NEG_5_PITCH is used to compute the super elevation of the pylon articulation in the CSU hydra_set_pylon_articulation.

```
/*
 * Get rocket range & calculate SuperElevation and Dispersion angles
 */
   pylons_set = FALSE;
   if( mun_data->data.rocket.articulation )
      range = weapons_get_rocket_range();
   else
      range = (REAL)(mun_data->data.rocket.flyout_range);
/*
 * Set pylon Super Elevation angle  &  pylon Dispersion angle
 */
    missile_hydra_set_pylon_articulation( range, warhead_class,
                              &flight_time,
                              &super_elev, &dispersion );
```

```
super_elev += HULL_NEG_5_PITCH;
rotate_set_angle( articulation(), super_elev );
rotate_set_angle( pylon_R(), (- dispersion) );
rotate_set_angle( pylon_L(), dispersion );
```

See APPENDIX N for a complete source code listing.

3.52.6 ARTICULATION_MAX

ARTICULATION_MAX is a constant defining the degress of maximum articulation.

3.52.6.1 Initialization

The constant is initialized during execution of the CSU hydra_init, called by CSC weapons_init. Execution of the CSU hydra_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define ARTICULATION_MAX        ( deg_to_rad(hydra_rkt_char[5]))
```

3.52.6.2 Usage

During real-time execution, this constant is not recomputed.

3.52.6.2.1 Algorithm

ARTICULATION_MAX is used to limit the initialization of the rotation element in the call to the CSU hydra_init.

```
if(!rotate_init_element( &articulation_element, hull(),
            1.0, 0.0, 0.0, 0.0,
            ARTICULATION_MIN,ARTICULATION_MAX,
        /*TWO_*/PI,
        /*rate*/
          0.0, HYDRA_LAUNCHER_POS_Y,
        HYDRA_LAUNCHER_POS_Z ))
```

See APPENDIX N for a complete source code listing.

3.53.1 M151_BURST_SPREAD

M151_BURST_SPREAD is a constant defining the radius of the M151 burst spread, especially, the M151 is twin bursts 3 meters apart.

3.53.1.1 Initialization

M151_BURST_SPREAD is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M151_BURST_SPREAD is also known as rkt_hydra_char[0].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,      /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,      /* release submunitions 180 ft  */
  M261_BURST_RANGE,       /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,      /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,       /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,      /* twin bursts are 35 m apart    */
  FLECH_60_MAX_RANGE,     /* darts fly total of 750 m     */
  50.0,          /* hydra minimum range */
 5000.0,         /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,         /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,         /* hydra maximum range for M261 */
 3200.0          /* hydra maximum range for M255  */
};
```

3.53.1.2 Usage

During real-time execution, this constant is not recomputed.

3.53.1.2.1 Algorithm

Rkt_hydra_char[0] is used to compute the lead_angle when the type of rocket is HE with 10 LB warhead by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_HE:                    /* type 10lb WARHEAD */
    if( range > HYDRA_MAX_RANGE_M151 )
       range = HYDRA_MAX_RANGE_M151;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                          ball_range, 0.0, 0.0,
                          time, se_angle );
   *lead_angle = atan( (rkt_hydra_char[ 0] - pylon_x) / range );
   *time = -5;      /* Does not have a timed fuze */
   break;
```

See APPENDIX M for a complete source code listing.

3.53.2 M261_BURST_HEIGHT

M261_BURST_HEIGHT is a constant defining the height of release for the M261 burst, especially, release of submunitions at 180 feet above the ground level.

3.53.2.1 Initialization

M261_BURST_HEIGHT is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M261_BURST_HEIGHT is also known as rkt_hydra_char[1].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart  */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart  */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart  */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m  */
  50.0,            /* hydra minimum range */
  5000.0,          /* hydra maximum range for Soviet S-5 57mm Rocket */
  7000.0,          /* hydra maximum range for _M151 [actual 9000 m] */
  7000.0,          /* hydra maximum range for M261 */
  3200.0           /* hydra maximum range for M255  */
};
```

3.53.2.2 Usage

During real-time execution, this constant is not recomputed.

3.53.2.2.1 Algorithm

Rkt_hydra_char[1] is used to compute ballistics trajectory when the rocket is a type MPSM by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_MPSM:                    /* type MPSM */
    if( range > HYDRA_MAX_RANGE_M261 )
       range = HYDRA_MAX_RANGE_M261;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                       ball_range, 0.0, rkt_hydra_char[ 1],
                       time, se_angle );
    *lead_angle = atan( (rkt_hydra_char[ 3] - pylon_x) / range );
   break;
```

Rkt_hydra_char[1] is assigned to submunition impact height when the rocket is a type MPSM by a call to the CSU missile_hydra_fire.

```
case ROCKET_MPSM:              /* Multi-Purpose Sub-Munition */
    bmptr->max_range = HYDRA_MAX_RANGE_M261;
    rkt->sub_mun_type = SUB_MUN_IMPACT;
    rkt->sub_ammo_type = munition_US_M73;
    rkt->sub_munition.impact.ammo = munition_US_M73;
    rkt->sub_munition.impact.fuze = munition_US_M433;
    rkt->sub_munition.impact.quantity = m73_per_m261_burst;
    rkt->sub_munition.impact.height = rkt_hydra_char[ 1];
    fuze = munition_US_M439;
  break;
```

See APPENDIX M for a complete source code listing.

3.53.3 M261_BURST_RANGE

M261_BURST_RANGE is a constant defining the distance from the target to the burst, especially, for the M261 burst at 0 meters in front of target.

3.53.3.1 Initialization

M261_BURST_RANGE is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M261_BURST_RANGE is also known as rkt_hydra_char[2].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart  */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart  */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart     */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m      */
  50.0,            /* hydra minimum range */
 5000.0,           /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,           /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,           /* hydra maximum range for M261 */
 3200.0            /* hydra maximum range for M255  */
};
```

3.53.3.2 Usage

During real-time execution, this constant is not recomputed.

3.53.3.2.1 Algorithm

Rkt_hydra_char[2] is not used in the current calculations.

See APPENDIX M for a complete source code listing.

3.53.4 M261_BURST_SPREAD

M261_BURST_SPREAD is a constant defining the radius of the M261 burst spread, especially, the M261 is twin bursts 13 meters apart.

3.53.4.1 Initialization

M261_BURST_SPREAD array is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M261_BURST_SPREAD is also known as rkt_hydra_char[3].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart    */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m      */
  50.0,             /* hydra minimum range */
 5000.0,            /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,            /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,            /* hydra maximum range for M261 */
 3200.0             /* hydra maximum range for M255  */
};
```

3.53.4.2 Usage

During real-time execution, this constant is not recomputed.

3.53.4.2.1 Algorithm

Rkt_hydra_char[3] is used to compute the lead angle when the rocket is a type MPSM by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_MPSM:                    /* type MPSM */
    if( range > HYDRA_MAX_RANGE_M261 )
       range = HYDRA_MAX_RANGE_M261;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                     ball_range, 0.0, rkt_hydra_char[ 1],
                     time, se_angle );
   *lead_angle = atan( (rkt_hydra_char[ 3] - pylon_x) / range );
   break;
```

See APPENDIX M for a complete source code listing.

3.53.5 M255_BURST_RANGE

M255_BURST_RANGE is a constant defining the distance from the target to the burst, especially, for the M255 burst at 150 meters in front of target.

3.53.5.1 Initialization

M255_BURST_RANGE is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M255_BURST_RANGE is also known as rkt_hydra_char[4].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart     */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m       */
  50.0,              /* hydra minimum range */
 5000.0,             /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,             /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,             /* hydra maximum range for M261 */
 3200.0              /* hydra maximum range for M255  */
};
```

3.53.5.2 Usage

During real-time execution, this constant is not recomputed.

3.53.5.2.1 Algorithm

Rkt_hydra_char[4] is used to compute ballistics trajectory and lead angle when the rocket is a type FLECHETTE by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_FLECHETTE:              /* type FLECHETTE */
    if( range > HYDRA_MAX_RANGE_M255 )
       range = HYDRA_MAX_RANGE_M255;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                         ball_range, rkt_hydra_char[ 4], 0.0,
                         time, se_angle );
    *lead_angle = atan((rkt_hydra_char[ 5] - pylon_x) /
                (range - rkt_hydra_char[ 4]));
   break;
```

See APPENDIX M for a complete source code listing.

3.53.6 M255_BURST_SPREAD

M255_BURST_SPREAD is a constant defining the radius of the M255 burst spread, especially, the M255 is twin bursts 35 meters apart.

3.53.6.1 Initialization

M255_BURST_SPREAD is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M255_BURST_SPREAD is also known as rkt_hydra_char[5].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,       /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,       /* release submunitions 180 ft  */
  M261_BURST_RANGE,        /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,       /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,        /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,       /* twin bursts are 35 m apart     */
  FLECH_60_MAX_RANGE,      /* darts fly total of 750 m       */
  50.0,            /* hydra minimum range */
 5000.0,           /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,           /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,           /* hydra maximum range for M261 */
 3200.0            /* hydra maximum range for M255  */
};
```

3.53.6.2 Usage

During real-time execution, this constant is not recomputed.

3.53.6.2.1 Algorithm

Rkt_hydra_char[5] is used to compute ballistics trajectory and lead angle when the rocket is a type FLECHETTE by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_FLECHETTE:                /* type FLECHETTE */
    if( range > HYDRA_MAX_RANGE_M255 )
      range = HYDRA_MAX_RANGE_M255;
   ball_range = range / speed_factor;
    missile_util_ballistics_calc_traj( ball_table, table_size,
                      ball_range, rkt_hydra_char[ 4], 0.0,
                      time, se_angle );
   *lead_angle = atan((rkt_hydra_char[ 5] - pylon_x) /
             (range - rkt_hydra_char[ 4]));
   break;
```

See APPENDIX M for a complete source code listing.

3.53.7 FLECH_60_MAX_RANGE

FLECH_60_MAX_RANGE is a constant defining the total distance the darts fly in meters after the proximity fuze detonates. At the maximum range, the flechette rounds have lost the momentum and fall to the ground.

3.53.7.1 Initialization

FLECH_60_MAX_RANGE is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. FLECH_60_MAX_RANGE is also known as rkt_hydra_char[6].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart    */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart      */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m       */
  50.0,            /* hydra minimum range */
 5000.0,           /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,           /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,           /* hydra maximum range for M261 */
 3200.0            /* hydra maximum range for M255  */
};
```

3.53.7.2 Usage

During real-time execution, this constant is not recomputed.

3.53.7.2.1 Algorithm

FLECH_60_MAX_RANGE is not used in the current calculations.

See APPENDIX M for a complete source code listing.

3.53.8 HYDRA_MIN_RANGE

HYDRA_MIN_RANGE is a constant defining the hydra minimum range.

3.53.8.1 Initialization

HYDRA_MIN_RANGE is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define  HYDRA_MIN_RANGE              rkt_hydra_char[ 7]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart    */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m      */
  50.0,            /* hydra minimum range */
  5000.0,          /* hydra maximum range for Soviet S-5 57mm Rocket */
  7000.0,          /* hydra maximum range for _M151 [actual 9000 m] */
  7000.0,          /* hydra maximum range for M261 */
  3200.0           /* hydra maximum range for M255  */
};
```

3.53.8.2 Usage

During real-time execution, this constant is not recomputed.

3.53.8.2.1 Algorithm

HYDRA_MIN_RANGE is used to limit the range to the target by a call to the CSU missile_hydra-set_pylon_articulation.

```
if( tgt_range < HYDRA_MIN_RANGE )
   range = HYDRA_MIN_RANGE;
else if(( max_range_limit > 0.0 )  &&
      ( tgt_range > max_range_limit ) )
   range = max_range_limit;
```

```
  else
    range = tgt_range;
```

See APPENDIX M for a complete source code listing.

3.53.9 HYDRA_MAX_RANGE_S5

HYDRA_MAX_RANGE_S5 is a constant defining the hydra maximum range for Soviet S-5 57mm rocket.

3.53.9.1 Initialization

HYDRA_MAX_RANGE_S5 is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define HYDRA_MAX_RANGE_S5   rkt_hydra_char[ 8]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart    */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft   */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart    */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart     */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m       */
  50.0,           /* hydra minimum range */
 5000.0,          /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,          /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,          /* hydra maximum range for M261 */
 3200.0           /* hydra maximum range for M255  */
};
```

3.53.9.2 Usage

During real-time execution, this constant is not recomputed.

3.53.9.2.1 Algorithm

HYDRA_MAX_RANGE_S5 is not used in the current calculations.

See APPENDIX M for a complete source code listing.

3.53.10 HYDRA_MAX_RANGE_M151

HYDRA_MAX_RANGE_M151 is a constant defining the hydra maximum range for the M151.

3.53.10.1 Initialization

HYDRA_MAX_RANGE_M151 array is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define  HYDRA_MAX_RANGE_M151       rkt_hydra_char[ 9]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,       /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,       /* release submunitions 180 ft  */
  M261_BURST_RANGE,        /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,       /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,        /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,       /* twin bursts are 35 m apart     */
  FLECH_60_MAX_RANGE,      /* darts fly total of 750 m       */
  50.0,            /* hydra minimum range */
 5000.0,           /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,           /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,           /* hydra maximum range for M261 */
 3200.0            /* hydra maximum range for M255  */
};
```

3.53.10.2 Usage

During real-time execution, this constant is not recomputed.

3.53.10.2.1 Algorithm

HYDRA_MAX_RANGE_M151 is used to bound the limit of the range for the individual rocket when the rocket type is HE by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_HE:                    /* type 10lb WARHEAD */
    if( range > HYDRA_MAX_RANGE_M151 )
       range = HYDRA_MAX_RANGE_M151;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                       ball_range, 0.0, 0.0,
                       time, se_angle );
    *lead_angle = atan( (rkt_hydra_char[ 0] - pylon_x) / range );
    *time = -5;       /* Does not have a timed fuze */
   break;
```

HYDRA_MAX_RANGE_M151 is assigned to the maximum range variable for the individual rocket when the rocket type is HE by a call to the CSU missile_hydra_fire.

```
case ROCKET_HE:                /* High Explosive */
    bmptr->max_range = HYDRA_MAX_RANGE_M151;
    rkt->sub_mun_type = SUB_MUN_NONE;
   rkt->sub_ammo_type = 0;
    fuze = munition_US_M433;
  break;
```

See APPENDIX M for a complete source code listing.

3.53.11 HYDRA_MAX_RANGE_M261

HYDRA_MAX_RANGE_M261 is a constant defining the hydra maximum range for the M261.

3.53.11.1 Initialization

HYDRA_MAX_RANGE_M261 is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define  HYDRA_MAX_RANGE_M261        rkt_hydra_char[10]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart    */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m      */
  50.0,             /* hydra minimum range */
  5000.0,           /* hydra maximum range for Soviet S-5 57mm Rocket */
  7000.0,           /* hydra maximum range for _M151 [actual 9000 m] */
  7000.0,           /* hydra maximum range for M261 */
  3200.0            /* hydra maximum range for M255  */
};
```

3.53.11.2 Usage

During real-time execution, this constant is not recomputed.

3.53.11.2.1 Algorithm

HYDRA_MAX_RANGE_M261 is used to bound the limit of the range for the individual rocket when the rocket type is MPSM by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_MPSM:                    /* type MPSM */
    if( range > HYDRA_MAX_RANGE_M261 )
       range = HYDRA_MAX_RANGE_M261;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                        ball_range, 0.0, rkt_hydra_char[ 1],
                        time, se_angle );
   *lead_angle = atan( (rkt_hydra_char[ 3] - pylon_x) / range );
   break;
```

HYDRA_MAX_RANGE_M261 is assigned to the maximum range variable for the individual rocket when the rocket type is MPSM by a call to the CSU missile_hydra_fire.

```
case ROCKET_MPSM:                /* Multi-Purpose Sub-Munition */
    bmptr->max_range = HYDRA_MAX_RANGE_M261;
    rkt->sub_mun_type = SUB_MUN_IMPACT;
    rkt->sub_ammo_type = munition_US_M73;
    rkt->sub_munition.impact.ammo = munition_US_M73;
    rkt->sub_munition.impact.fuze = munition_US_M433;
    rkt->sub_munition.impact.quantity = m73_per_m261_burst;
    rkt->sub_munition.impact.height = rkt_hydra_char[ 1];
    fuze = munition_US_M439;
  break;
```

See APPENDIX M for a complete source code listing.

3.53.12 HYDRA_MAX_RANGE_M255

HYDRA_MAX_RANGE_M255 is a constant defining the hydra maximum range for the M255.

3.53.12.1 Initialization

HYDRA_MAX_RANGE_M255 is initialized during execution of the CSU missile_hydra_init, called by CSU hydra_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define  HYDRA_MAX_RANGE_M255        rkt_hydra_char[11]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,        /* twin bursts are 3 m apart   */
  M261_BURST_HEIGHT,        /* release submunitions 180 ft  */
  M261_BURST_RANGE,         /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,        /* twin bursts are 13 m apart   */
  M255_BURST_RANGE,         /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,        /* twin bursts are 35 m apart    */
  FLECH_60_MAX_RANGE,       /* darts fly total of 750 m      */
  50.0,            /* hydra minimum range */
 5000.0,           /* hydra maximum range for Soviet S-5 57mm Rocket */
 7000.0,           /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,           /* hydra maximum range for M261 */
 3200.0            /* hydra maximum range for M255  */
};
```

3.53.12.2 Usage

During real-time execution, this constant is not recomputed.

3.53.12.2.1 Algorithm

HYDRA_MAX_RANGE_M255 is used to bound the limit of the range for the individual rocket when the rocket type is FLECHETTE by a call to the CSU missile_hydra_set_pylon_articulation.

```
case ROCKET_FLECHETTE:                /* type FLECHETTE */
    if( range > HYDRA_MAX_RANGE_M255 )
       range = HYDRA_MAX_RANGE_M255;
   ball_range = range / speed_factor;
     missile_util_ballistics_calc_traj( ball_table, table_size,
                         ball_range, rkt_hydra_char[ 4], 0.0,
                         time, se_angle );
    *lead_angle = atan((rkt_hydra_char[ 5] - pylon_x) /
              (range - rkt_hydra_char[ 4]));
   break;
```

HYDRA_MAX_RANGE_M255 is assigned to the maximum range variable for the individual rocket when the rocket type is FLECHETTE by a call to the CSU missile_hydra_fire.

```
case ROCKET_FLECHETTE:            /* Flechette discharging warhead */
    bmptr->max_range = HYDRA_MAX_RANGE_M255;
    rkt->sub_mun_type = SUB_MUN_CANISTER;
    rkt->sub_ammo_type = munition_US_Flechette_60;
    rkt->sub_munition.dart.ammo = munition_US_Flechette_60;
   rkt->sub_munition.dart.fuze = 0;
    fuze = munition_US_M439;
  break;
```

See APPENDIX M for a complete source code listing.

3.54 Sub_m73_char

The sub_m73_char array consists of characteristics and parameters describing a M73 submunition flyout.

3.54.1 Sub_m73_char[0]

Sub_m73_char[0] is a constant defining the downward acceleration of gravity as 75% of gravity, per tick squared (75% * (9.8m/sec**2)/225 ticks**2).

3.54.1.1 Initialization

The sub_m73_char[0] constant is initialized during execution of the CSU missile_m73_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.54. - Submunitions M73 Characteristics Data Array for a summary of the array data.

The following declaration is for the default values.

```
static REAL sub_m73_char[3] =
{
 0.03266667,          /* 75% of gravity - 75% * 9.8m/sec^^2/225 ticks^^2*/
  M73_FOOT_ANGLE_X,          /* bomblettes fall w/ +/- 8.8 deg angular displ
*/
  M73_FOOT_ANGLE_Y  /* bomblettes fall w/ +/- 12.35 deg angular displ */
};
```

3.54.1.2 Usage

During real-time execution, this constant is not recomputed.

3.54.1.2.1 Algorithm

Sub_m73_char[0] is used to compute the impact timer for the for the M73 by a call to the CSU missile_m73_drop.

```
impact = &(sub_mun->impact);
if( impact->timer == 0 )
{
    if( missile_util_comm_check_sub_mun( bmptr,
                                         MSL_TYPE_BALLISTIC,
                                         sub_mun, SUB_MUN_IMPACT ))
  {
    if( impact->distance > 0.0 )
       impact->timer = (int)
          ((8 * scaled_rand()) + 1.0 +
           (sqrt((1.9 * impact->distance) / sub_m73_char[0])));
    else
       impact->timer = -1;
  }
  else
  {
    impact_pt[X] = bmptr->location[X];
    impact_pt[Y] = bmptr->location[Y] - 10;
    if( traj_up )
       impact_pt[Z] = bmptr->location[Z] + impact->distance;
    else
       impact_pt[Z] = 10;
    traj_up = ( ! traj_up );
       missile_util_comm_release_sub_munition( bmptr,
                                         MSL_TYPE_BALLISTIC,
                                         sub_mun, SUB_MUN_IMPACT,
                                         impact_pt, zero_velocity );
  }
  return( FALSE );
}
else
{
   if( bmptr->time < impact->timer )       /* wait until sub_mun's */
  {                            /* hit the ground....  */
    bmptr->time += 1;                /* incr time counter   */
    return( FALSE );
  }
  else                          /*  ie. time == timer */
  {
```

```
        if( impact->timer > 0 )
        {
            missile_m73_get_impact( bmptr->location, impact_pt,
                        bmptr->launcher_C_world,
                        impact->distance );
              missile_util_comm_release_sub_munition
              ( bmptr, MSL_TYPE_BALLISTIC, sub_mun,
               SUB_MUN_IMPACT, impact_pt, zero_velocity );
        }
/* reset time counter */
        bmptr->time = 0;
        return( TRUE );
      }
    }
```

See APPENDIX O for a complete source code listing.

3.54.2 M73_FOOT_ANGLE_X

M73_FOOT_ANGLE_X is a constant defining the dispersion angle on the x-axis of bomblettes as they fall, especially, falling with +/- 8.8 degrees angular displacement along the x-axis.

3.54.2.1 Initialization

M73_FOOT_ANGLE_X is initialized during execution of the CSU missile_m73_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.54. - Submunitions M73 Characteristics Data Array for a summary of the constant. M73_FOOT_ANGLE_X is also known as sub_m73_char[2].

The following declaration is for the default values.

```
static REAL sub_m73_char[3] =
{
 0.03266667,        /* 75% of gravity - 75% * 9.8m/sec^^2/225 ticks^^2*/
  M73_FOOT_ANGLE_X,        /* bomblettes fall w/ +/- 8.8 deg angular displ
*/
  M73_FOOT_ANGLE_Y  /* bomblettes fall w/ +/- 12.35 deg angular displ */
};
```

3.54.2.2 Usage

During real-time execution, this constant is not recomputed.

3.54.2.2.1 Algorithm

Sub_m73_char[2] is used to compute the detonation point in CSU missile_m73_get_impact.

```
x = height * sin(deg_to_rad( sub_m73_char[1] * (0.50 - scaled_rand())));
y = height * sin(deg_to_rad( sub_m73_char[2] * (0.50 - scaled_rand())));
detonation[X] = x * mCw[0][0] - y * mCw[0][1];
detonation[Y] = y * mCw[0][0] + x * mCw[0][1];
detonation[Z] = - height;
```

See APPENDIX O for a complete source code listing.

3.54.3 M73_FOOT_ANGLE_Y

M73_FOOT_ANGLE_Y is a constant defining the dispersion angle on the y-axis of bomblettes as they fall, especially, falling with +/- 12.35 degrees angular displacement along the y-axis.

3.54.3.1 Initialization

M73_FOOT_ANGLE_Y is initialized during execution of the CSU missile_m73_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.54. - Submunitions M73 Characteristics Data Array for a summary of the constant. M73_FOOT_ANGLE_Y is also known as sub_m73_char[3].

The following declaration is for the default values.

```
static REAL sub_m73_char[3] =
{
 0.03266667,        /* 75% of gravity - 75% * 9.8m/sec^^2/225 ticks^^2*/
  M73_FOOT_ANGLE_X,        /* bomblettes fall w/ +/- 8.8 deg angular displ
*/
  M73_FOOT_ANGLE_Y  /* bomblettes fall w/ +/- 12.35 deg angular displ */
};
```

3.54.3.2 Usage

During real-time execution, this constant is not recomputed.

3.54.3.2.1 Algorithm

Sub_m73_char[3] is used to compute the detonation point in CSU missile_m73_get_impact.

```
x = height * sin(deg_to_rad( sub_m73_char[1] * (0.50 - scaled_rand())));
y = height * sin(deg_to_rad( sub_m73_char[2] * (0.50 - scaled_rand())));
detonation[X] = x * mCw[0][0] - y * mCw[0][1];
detonation[Y] = y * mCw[0][0] + x * mCw[0][1];
detonation[Z] = - height;
```

See APPENDIX O for a complete source code listing.

3.55 Sub_flech_char

The sub_flech_char array consists of characteristics and parameters describing M73 bomblettes falling.

3.55.1 INVEST_DIST_SQ

The INVEST_DIST_SQ is a constant defining the area at a maximum speed of less than 100 m/sec.

3.55.1.1 Initialization

The INVEST_DIST_SQ is initialized during execution of the CSU missile_flechette_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.55. - Submunitions Flechette Characteristics Data Array for a summary of the constant.

```
#define INVEST_DIST_SQ          sub_flech_char[0]
```

The following declaration is for the default values.

```
static REAL sub_flech_char[3] =
{
 10000.0,  /* (100 m)^2 :: max speed < 100 */
  306.25,  /* (17.5 m)^2 :: flechettes fly
                  in a cylinder with a radius
                  of 17.5 m and length of 750 m */
 FLECH_60_MAX_RANGE       /* darts fly total of 750m */
};
```

3.55.1.2 Usage

During real-time execution, this constant is not recomputed.

3.55.1.2.1 Algorithm

INVEST_DIST_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile_flechette_fly.

```
* PROX_FUZE */
   if( missile_fuze_all_prox(. bmptr,
                    MSL_TYPE_BALLISTIC, PROX_FUZE_ON_ALL_VEH,
                   &(null_VehicleID), &(dart->pptr),
                    veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ ) )
     do
     {
/* DETONATION ? */
          if( missile_util_comm_check_sub_mun( bmptr,
                         MSL_TYPE_BALLISTIC,
                         sub_mun,  SUB_MUN_CANISTER  ))
            missile_util_comm_release_sub_munition( bmptr,
                              MSL_TYPE_BALLISTIC,
                              sub_mun,
                               SUB_MUN_CANISTER,
                              zero_vector,
                              velocity );
     } while( dart->pptr != NULL  &&
            missile_fuze_detonate_prox( bmptr, MSL_TYPE_BALLISTIC,
                         &(dart->pptr), FUZE_DIST_SQ, 0 ));
```

See APPENDIX P for a complete source code listing.

3.55.2 FUZE_DIST_SQ

FUZE_DIST_SQ is a constant defining the square of the radius of the cylinder describing the flechettes fly in a cylinder with a radius of 17.5 meters and a length of 750 meters

3.55.2.1 Initialization

The FUZE_DIST_SQ is initialized during execution of the CSU missile_flechette_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.55. - Submunitions Flechette Characteristics Data Array for a summary of the constant.

```
#define FUZE_DIST_SQ          sub_flech_char[1]
```

The following declaration is for the default values.

```
static REAL sub_flech_char[3] =
{
 10000.0,  /* (100 m)^2  :: max speed < 100 */
   306.25,  /* (17.5 m)^2 :: flechettes fly
                   in a cylinder with a radius
                   of 17.5 m and length of 750 m */
 FLECH_60_MAX_RANGE       /* darts fly total of 750m */
};
```

3.55.2.2 Usage

During real-time execution, this constant is not recomputed.

3.55.2.2.1 Algorithm

FUZE_DIST_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile_flechette_fly.

```
* PROX_FUZE */
  if( missile_fuze_all_prox( bmptr,
                    MSL_TYPE_BALLISTIC, PROX_FUZE_ON_ALL_VEH,
                   &(null_VehicleID), &(dart->pptr),
                    veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ ) )
    do
    {
/* DETONATION ? */
        if( missile_util_comm_check_sub_mun( bmptr,
                          MSL_TYPE_BALLISTIC,
                          sub_mun, SUB_MUN_CANISTER ))
           missile_util_comm_release_sub_munition( bmptr,
                                MSL_TYPE_BALLISTIC,
                                sub_mun,
                                 SUB_MUN_CANISTER,
                                zero_vector,
                                velocity );
    } while( dart->pptr != NULL &&
           missile_fuze_detonate_prox( bmptr, MSL_TYPE_BALLISTIC,
                        &(dart->pptr), FUZE_DIST_SQ, 0 ));
```

See APPENDIX P for a complete source code listing.

3.55.3 FLECH_60_MAX_RANGE

FLECH_60_MAX_RANGE is a constant defining the total distance the darts fly in meters after the proximity fuze detonates. At the maximum range, the flechette rounds have lost the momentum and fall to the ground. This constant is also known as sub_flech_char[2].

3.55.3.1 Initialization

The constant FLECH_60_MAX_RANGE is initialized during execution of the CSU missile_flechette_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.55. - Submunitions Flechette Characteristics Data Array for a summary of the constant.

The following declaration is for the default values.

```
static REAL sub_flech_char[3] =
{
 10000.0,  /* (100 m)^2  :: max speed < 100 */
   306.25,  /* (17.5 m)^2 :: flechettes fly
                    in a cylinder with a radius
                    of 17.5 m and length of 750 m */
  FLECH_60_MAX_RANGE       /* darts fly total of 750m */
};
```

3.55.3.2 Usage

During real-time execution, this constant is not recomputed.

3.55.3.2.1 Algorithm

FLECH_60_MAX_RANGE is used to compute the termination of the flight for the canister and darts.

```
  dart->distance += bmptr->speed;
  if( dart->distance >= sub_flech_char[2] )
     return( FALSE );
```

See APPENDIX P for a complete source code listing.

3.56 Flechette_speed_coef

The flechette_speed_coef array consists of the coefficients for a polynomial equation defining the flechette flyout speed with respect to time in the form using the Newton-Raphson method.

3.56.1 Initialization

The flechette_speed_coef array is initialized during execution of the CSU missile_flechette_init, called by CSU missile_hydra_fly_rockets. See TABLE 5.1.56. - Flechette Speed Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

3.56.2 Usage

During real-time execution, this array is not recomputed. FLECHETTE_SPEED_DEG determines the number of elements of the array to be used in the polynomial evaluation.

3.56.2.1 Algorithm

The flechette_speed_coef array is used to compute the flechette speed during free fall using the CSU missile_util_eval_poly, and called by the CSU missile_flechette_fly. The CSU missile_util_eval_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and distance.

```
bmptr->speed =
    missile_util_eval_poly( FLECHETTE_SPEED_DEG,
                            flechette_speed_coef,
                            dart->distance ) + dart->init_speed;
```

See APPENDIX P for a complete source code listing.

4. Error messages.

The error messages are located in the source code. See the appropriate Appendix for the error messages.

5. CSCI data.

This section describes only those global data elements modified or added within the CSCI under this delivery order. For ease in readability and maintenance, the information is provided in tables.

5.1. Data elements internal to the CSCI.

a. For data elements internal to the CSCI, the following tables describe the data arrays and the data.

TABLE 5.1. - SUMMARY of DATA ARRAYS

NAME of DATA ARRAY	DESCRIPTION [NOTE 1]	SIZE of ARRAY	DATA TYPE [NOTE 2]	FREQUENCY of CALCULATION	DECLARATION/DEFAULT MODULE	DATA SOURCE
aero_data	This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control.	100	REAL	15 Hz	rwa_aerodyn.c	simnet/data/rwa_aero.d
aero_init	This data array consists of initial values for positions of the control inputs, stabilator augementation integrators, attitude control integrators, and hover aumentation integrators.	20	REAL	15 Hz	rwa_aerodyn.c	simnet/data/rw_ae_in.d
aero_simple	This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "simple" mode.	20	REAL	15 Hz	rwa_aerodyn.c	simnet/data/rw_ae_sp.d
aero_stealth	This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "stealth" mode.	20	REAL	15 Hz	rwa_aerodyn.c	simnet/data/rw_ae_st.d
engine_data	This data array consists of characteristics and parameters describing the engine performance and control.	20	REAL	15 Hz	rwa_engine.c	simnet/data/rwa_engn.d
engine_init_data	This data array consists of initial values of the current engine state, performance, and control.	10	REAL	15 Hz	rwa_engine.c	simnet/data/rw_en_in.d
engine_stat_data	This data array consists of the initial values for flight time, engine status, number of engines, and powertrain damage status.	10	int	15 Hz	rwa_engine.c	simnet/data/rw_en_st.d
kinemat_data	This data array consists of kinematic constants and limits for the vehicle and its control.	20	REAL	15 Hz	rwa_kinemat.c	simnet/data/rwa_kine.d
kinemat_init_data	This data array consists of initial values for kinematic variables including velocity, angle-of-attack, pitch, altitude, heading, and g-force.	30	REAL	15 Hz	rwa_kinemat.c	simnet/data/rw_ki_in.d
adat_miss_char	This data array consists of characteristics and parameters describing an ADAT missile system and its performance constraints.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_.chd
adat_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, maximum cosines of turns while unpowered, and temporal bias for the ADAT missile.	5	int	15 Hz	miss_atad.c	See DESCRIPTION of individual elements of TABLE
adat_burn_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile burn speed with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_bs.d
adat_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile coast speed with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_cs.d
adat_burn_turn_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile maximum cosine of turn while powered with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_bt.d
adat_coast_turn_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile maximum cosine of turn while unpowered with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_ct.d

TABLE 5.1. - SUMMARY of DATA ARRAYS
[Continued]

NAME of DATA ARRAY	DESCRIPTION [NOTE 1]	SIZE of ARRAY	DATA TYPE [NOTE 2]	FREQUENCY of CALCULATION	DECLARATION/DEFAULT MODULE	DATA SOURCE
adat_temp_bias_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile temporal bias with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_tb.d
hellfr_miss_char	This data array consists of characteristics and parameters describing a Hellfire missile system and its performance constraints.	15	REAL	15 Hz	miss_hellfr.c	simnet/data/ms_hf_ch.d
hellfr_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the time-of-flight, the burn speed, and the coast speed for the Hellfire missile.	3	int	15 Hz	miss_hellfr.c	See DESCRIPTION of individual elements of TABLE
hellfire_tof_coeff	This data array consists of the coefficients for a polynomial equation defining the Hellfire missile time-of-flight with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_hellfr.c	simnet/data/ms_hf_tf.d
hellfire_burn_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Hellfire missile burn speed with repect to range in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_hellfr.c	simnet/data/ms_hf_bs.d
hellfire_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Hellfire missile coast speed with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_hellfr.c	simnet/data/ms_hf_cs.d
kem_miss_char	This data array consists of characteristics and parameters describing a KEM missile system and its performance constraints.	10	REAL	15 Hz	misskem.c	simnet/data/ms_km_ch.d
kem_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the KEM missile.	5	int	15 Hz	misskem.c	See DESCRIPTION of individual elements of TABLE
kem_burn_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the KEM missile burn speed with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	misskem.c	simnet/data/ms_km_bs.d
kem_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the KEM missile coast speed with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	misskem.c	simnet/data/ms_km_cs.d
kem_burn_turn_coeff	This data array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of turn while powered with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	misskem.c	simnet/data/ms_km_bt.d
kem_coast_turn_coeff	This data array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of turn while unpowered with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	misskem.c	simnet/data/ms_km_ct.d
maverick_miss_char	This data array consists of characteristics and parameters describing a Maverick missile system and its performance constraints.	15	REAL	15 Hz	miss_maverck.c	simnet/data/ms_mk_ch.d

TABLE 5.1. - SUMMARY of DATA ARRAYS
[Continued]

NAME of DATA ARRAY	DESCRIPTION [NOTE 1]	SIZE of ARRAY	DATA TYPE [NOTE 2]	FREQUENCY of CALCULATION	DECLARATION/DEFAULT MODULE	DATA SOURCE
maverick_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Maverick missile.	2	int	15 Hz	miss_maverck.c	See DESCRIPTION of individual elements of TABLE
maverick_burn_speed_coeff_	This data array consists of the coefficients for a polynomial equation defining the Maverick missile burn speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss_maverck.c	simnet/data/ms_mk_bs.d
maverick_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Maverick missile coast speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss_maverck.c	simnet/data/ms_mk_cs.d
nlos_miss_char	This data array consists of characteristics and parameters describing a NLOS missile system and its performance constraints.	20	REAL	15 Hz	miss-nlos.c	simnet/data/ms_nl_ch.d
nlos_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the time-of-flight, the burn speed, and the coast speed for the NLOS missile.	5	int	15 Hz	miss-nlos.c	See DESCRIPTION of individual elements of TABLE
nlos_burn_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the NLOS missile burn speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss-nlos.c	simnet/data/ms_nl_bs.d
nlos_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the NLOS missile coast speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss-nlos.c	simnet/data/ms_nl_cs.d
stinger_miss_char	This data array consists of characteristics and parameters describing a Stinger missile system and its performance constraints.	15	REAL	15 Hz	miss_stinger.c	simnet/data/ms_st_ch.d
stinger_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Stinger missile.	2	int	15 Hz	miss_stinger.c	See DESCRIPTION of individual elements of TABLE
stinger_burn_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Stinger missile burn speed with repect to time in the form using the Newton-Raphson method.	2	REAL	15 Hz	miss_stinger.c	simnet/data/ms_st_bs.d
stinger_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Stinger missile coast speed with repect to time in the form using the Newton-Raphson method.	4	REAL	15 Hz	miss_stinger.c	simnet/data/ms_st_cs.d
tow_miss_char	This data array consists of characteristics and parameters describing a TOW missile system and its performance constraints.	5	REAL	15 Hz	miss_tow.c	simnet/data/ms_tw_ch.d
tow_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the TOW missile.	5	int	15 Hz	miss_tow.c	See DESCRIPTION of individual elements of TABLE
tow_burn_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the TOW missile burn speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss_tow.c	simnet/data/ms_tw_bs.d

TABLE 5.1. - SUMMARY of DATA ARRAYS
[Continued]

NAME of DATA ARRAY	DESCRIPTION	SIZE of ARRAY	DATA TYPE [NOTE 1]	FREQUENCY of CALCULATION	DECLARATION/DEFAULT MODULE	DATA SOURCE
tow_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the TOW missile coast speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss_tow.c	simnet/data/ms_tw_cs.d
tow_burn_turn_coeff	This two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the TOW missile maximum cosine of turn while powered with repect to time in the form using the Newton-Raphson method.	3 x 2	MAX_COS_COEFF	15 Hz	miss_tow.c	simnet/data/ms_tw_bt.d
tow_coast _turn_coeff	This two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the TOW missile maximum cosine of turn while unpowered with repect to time in the form using the Newton-Raphson method.	3 x 4	MAX_COS_COEFF	15 Hz	miss_tow.c	simnet/data/ms_tw_ct.d
tow_miss_char [NOTE 3]	This data array consists of characteristics and parameters describing an ATGM missile system and its performance constraints.	5	REAL	15 Hz	miss_atgm.c	simnet/data/ms_at_ch.d
tow_miss_poly_deg [NOTE 3]	This data array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the ATGM missile.	5	int	15 Hz	miss_atgm.c	See DESCRIPTION of individual elements of TABLE
tow_burn_speed_coeff [NOTE 3]	This data array consists of the coefficients for a polynomial equation defining the ATGM missile burn speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss_atgm.c	simnet/data/ms_at_bs.d
tow_coast_speed_coeff [NOTE 3]	This data array consists of the coefficients for a polynomial equation defining the ATGM missile coast speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	miss_atgm.c	simnet/data/ms_at_cs.d
tow_burn_turn_coeff [NOTE 3]	This two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM missile maximum cosine of turn while powered with repect to time in the form using the Newton-Raphson method.	3 x 2	MAX_COS_COEFF	15 Hz	miss_atgm.c	simnet/data/ms_at_bt.d
tow_coast _turn_coeff [NOTE 3]	This two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM missile maximum cosine of turn while unpowered with repect to time in the form using the Newton-Raphson method.	3 x 4	MAX_COS_COEFF	15 Hz	miss_atgm.c	simnet/data/ms_at_ct.d
rkt_hydra_char	This data array consists of characteristics and parameters describing a Hyra 70 missile burst.	12	REAL	15 Hz	rkt_hydra.c	simnet/data/rkt_hydr.d
hydra_rkt_char	This data array consists of characteristics and parameters describing a missile launcher system and its performance constraints.	7	REAL	15 Hz	rwa_hydra.c	simnet/data/rwa_hydr.d

TABLE 5.1. - SUMMARY of DATA ARRAYS
[Continued]

NAME of DATA ARRAY	DESCRIPTION	SIZE of ARRAY	DATA TYPE (NOTE 1)	FREQUENCY of CALCULATION	DECLARATION/DEFAULT MODULE	DATA SOURCE
sub_flech_char	This data array consists of characteristics and parameters describing a flechette flyout.	3	REAL	15 Hz	sub_flech.c	simnet/data/sub_flec.d
flechette_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the flechette flyout speed with repect to time in the form using the Newton-Raphson method.	5	REAL	15 Hz	sub_flech.c	simnet/data/flec_spd.d
sub_m73_char	This data array consists of characteristics and parameters describing M73 bomblettes falling.	3	REAL	15 Hz	sub_m73.c	simnet/data/sub_m73.d

NOTE 1 See individual TABLES for description of individual elements.

NOTE 2 int is a "C" type for integer.
REAL is a "C" macro DEFINE for type float.
MAX_COS_COEFF is a "C" macro DEFINE for a structure of REAL types.

NOTE 3 The ATGM missile module uses the same data array names as the TOW missile module. The function names have been changed to reflect ATGM.. The modules are used in separate builds.

TABLE 5.1.1. - AERODYNAMICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_data[0]	MOMENT_OF_INERTIA_X;	rad/sec	50000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[1]	MOMENT_OF_INERTIA_Y;	kg-m**2	50000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[2]	MOMENT_OF_INERTIA_Z;	kg-m**2	50000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[3]	AIRFRAME_MASS;	kg	4881.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[4]	ORDINANCE_MASS;	kg	1591.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[5]	GRAV_CONSTANT;	m/sec**2	9.8	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[6]	CG_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[7]	CG_AC_Y;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[8]	CG_AC_Z;		-0.1	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[9]	VIRTUAL_WING_AREA;	m**2	25.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[10]	VIRTUAL_WING_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[11]	VIRTUAL_WING_COP_AC_Y;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[12]	VIRTUAL_WING_COP_AC_Z;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[13]	WING_LIFT_COEFFICIENT_FIT_3;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[14]	WING_LIFT_COEFFICIENT_FIT_2;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[15]	WING_LIFT_COEFFICIENT_FIT_1;		1.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[16]	WING_LIFT_COEFFICIENT_FIT_0;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[17]	WING_STALL_AOA;	deg	30.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[18]	VSTAB_AREA;	m**2	30	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[19]	VSTAB_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[20]	VSTAB_COP_AC_Y;		-9.1	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[21]	VSTAB_COP_AC_Z;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[22]	VSTAB_LIFT_COEFFICIENT_1;		5.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[23]	VSTAB_STALL_SSA;	deg	60.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[24]	MAIN_ROTOR_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[25]	MAIN_ROTOR_COP_AC_Y;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[26]	MAIN_ROTOR_COP_AC_Z;		2.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[27]	MAIN_ROTOR_MAX_THRUST;	N	123500.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d

TABLE 5.1.1. - AERODYNAMICS DATA ARRAY
[Continued]

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_data[28]	MAIN_ROTOR_MAST_TILT;	deg	2.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[29]	MAIN_ROTOR_MAX_LOAD_TORQUE;	N-m	76476.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[30]	MAIN_ROTOR_MAX_PITCH_MOMENT;	N-m	100000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[31]	MAIN_ROTOR_MAX_ROLL_MOMENT;	N-m	100000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[32]	MAIN_ROTOR_TORQUE_COUPLING_GAIN;		0.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[33]	MAIN_ROTOR_GROUND_EFFECT_FACTOR;		0.4	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[34]	TAIL_ROTOR_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[35]	TAIL_ROTOR_COP_AC_Y;		-9.1	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[36]	TAIL_ROTOR_COP_AC_Z;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[37]	TAIL_ROTOR_MAX_THRUST;	N	8909.1	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[38]	TAIL_ROTOR_MAX_LOAD_TORQUE;	N-m	1684.8	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[39]	P_DRAG_COEFF_CONST;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[40]	P_DRAG_TAS_BREAK;		50.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[41]	P_DRAG_COEFF_BREAK;		0.02	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[42]	P_DRAG_TAS_MAX;		100.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[43]	P_DRAG_COEFF_MAX;		0.06	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[44]	TOTAL_WETTED_SURFACE_AREA;		50.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[45]	MAX_ATT_CTL_ANGLE_STOP;	deg	6.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[46]	MAX_ATT_DAMPING_FACTOR;		4.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[47]	HOVER_SLOW_LIMIT;		5.15	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[48]	HOVER_AUG_PITCH_RESET_VALUE;		0.044	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[49]	MAX_ATT_CTL_ANGLE_NORM;	deg	15.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[50]	MAX_ATT_CTL_ANGLE_MED;	deg	10.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[51]	MAX_ATT_CTL_ANGLE_SLOW;	deg	6.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[52]	HOVER_MED_LIMIT;		15.46	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[53]	ATT_CTL_PITCH_P_GAIN;		2.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[54]	ATT_CTL_PITCH_I_GAIN;		0.05	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d

TABLE 5.1.1. - AERODYNAMICS DATA ARRAY
[Continued]

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_data[55]	ATT_CTL_ROLL_P_GAIN;		5.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[56]	ATT_CTL_ROLL_I_GAIN;		0.05	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[57]	HOVER_AUG_ROLL_P_GAIN;		0.100	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[58]	HOVER_AUG_ROLL_I_GAIN;		0.001	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[59]	HOVER_AUG_PITCH_P_GAIN;		0.100	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[60]	HOVER_AUG_PITCH_I_GAIN;		0.001	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[61]	HOVER_AUG_YAW_P_GAIN;		10.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[62]	HOVER_AUG_YAW_I_GAIN;		5.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[63]	HOVER_AUG_CLIMB_P_GAIN;		1.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[64]	HOVER_AUG_CLIMB_I_GAIN;		0.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[65]	MAX_STAB_AUG_PITCH_ROLL_CONTROL;		0.20	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[66]	MAX_STAB_AUG_YAW_CLIMB_CONTROL;		0.05	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[67]	ROLL_RATE_DAMPING_GAIN;		100000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[68]	PITCH_RATE_DAMPING_GAIN;		100000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[69]	YAW_RATE_DAMPING_GAIN;		100000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[70]	VERTICAL_RATE_DAMPING_GAIN;		2000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[71]	LATERAL_VELOCITY_DAMPING_GAIN;		1000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[72]	LIFT_COEFF_VIRTUAL_WING;		0.6	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[73]	OSWALD_EFFIC_FACTOR;		0.9	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[74]	INDUCED_DRAG_COEFF;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[75]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[76]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[77]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[78]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[79]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[80]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[81]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d

TABLE 5.1.1. - AERODYNAMICS DATA ARRAY
[Continued]

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_data[82]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[83]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[84]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[85]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[86]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[87]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[88]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[89]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[90]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[91]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[92]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[93]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[94]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[95]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[96]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[97]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[98]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[99]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.2. - AERODYNAMICS INITIALIZATION DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_init[0]	cyclic pitch		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[1]	cyclic roll		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_init	simnet/data/rw_ae_in.d
aero_init[2]	collective		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_init	simnet/data/rw_ae_in.d
aero_init[3]	pedal		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[4]	stab aug pitch integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[5]	stab aug roll integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[6]	stab aug yaw integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[7]	stab aug climb integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[8]	attitude control pitch integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[9]	attitude control roll integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[10]	hover aug pitch integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[11]	hover aug roll integrator		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[12]	hover aug pitch angle		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[13]	hover aug roll angle		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[14]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_in.d
aero_init[15]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_in.d
aero_init[16]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_in.d
aero_init[17]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_in.d
aero_init[18]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_in.d
aero_init[19]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_in.d

NOTE 1 . REAL is a "C" macro DEFINE for type float.

TABLE 5.1.3. - AERODYNAMICS SIMPLE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_simple[0]	MAX_HELICOPTER_POWER;	N	500000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[1]	MAX_HH;	rad	0.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_init	simnet/data/rw_ae_sp.d
aero_simple[2]	H_K1; gain on position error		48.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_init	simnet/data/rw_ae_sp.d
aero_simple[3]	H_K2; gain on gravity term of power setting		0.15	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[4]	H_K7; air drag coefficient		10.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[5]	H_K8; air drag coefficient	kg	100.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[6]	H_KP; power		150000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[7]	H_KPR; pitch/roll constant, approximately pi/3		1.5	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[8]	H_KY; yaw constant, approximately pi/2		0.7	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[9]	H_KH; hover hold gain on velocity term		0.03	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[10]	H_CHH; collective hover hold gain		400000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[11]	H_CL; coefficient of lift		100.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[12]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[13]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[14]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[15]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[16]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[17]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[18]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d
aero_simple[19]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_init		simnet/data/rw_ae_sp.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.4. - AERODYNAMICS STEALTH DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
aero_stealth[0]	H_FWD_MUL;		80.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
aero_stealth[1]	H_SIDE_MUL;		30.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_stealth	simnet/data/rw_ae_sl.d
aero_stealth[2]	H_COLL_MUL;		10.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_stealth	simnet/data/rw_ae_sl.d
aero_stealth[3]	MAX_TORQUE;		10000000000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
aero_stealth[4]	MAX_FORCE;		10000000000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
aero_stealth[5]	MASS;	kg	5000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
aero_stealth[6]	INERTIA;		25000.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
aero_stealth[7]	DEAD_ZONE;		0.03	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
aero_stealth[8]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[9]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[10]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[11]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[12]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[13]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[14]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[15]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[16]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[17]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[18]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d
aero_stealth[19]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simnet/data/rw_ae_sl.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.5. - ENGINE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
engine_data[0]	GOVERNOR_ENGINE_SPEED_SETTING;	rad/sec	1030.55	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[1]	GOVERNOR_P_GAIN;		0.05	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_init	simnet/data/rwa_engn.d
engine_data[2]	GOVERNOR_I_GAIN;		0.05	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_init	simnet/data/rwa_engn.d
engine_data[3]	MAX_ENGINE_TORQUE;	N-m	1031.6	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[4]	MIN_ENGINE_LOAD_TORQUE;	N-m	25.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[5]	MAX_ENGINE_PERCENT_POWER;	percent	1.2	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[6]	ENGINE_TORQUE_INTERCEPT;		1200.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[7]	ENGINE_TORQUE_SLOPE;		0.16438	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[8]	NOSE_GEARBOX_RATIO;		2.130	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[9]	MAIN_ROTOR_GEAR_RATIO;		34.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[10]	TAIL_ROTOR_GEAR_RATIO;		7.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[11]	POWERTRAIN_INERTIA;		100.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[12]	MAX_FUELFLOW;	gals/hr	153.8461539	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[13]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[14]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[15]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[16]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[17]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[18]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[19]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.6. - ENGINE INITIALIZATION DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
engine_init_data[0]	engine power		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_init_data[1]	engine percent torque		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_init	simnet/data/rwa_engn.d
engine_init_data[2]	engine speed		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_init	simnet/data/rwa_engn.d
engine_init_data[3]	integrator gain		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_init_data[4]	last percent shaft speed		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_init_data[5]	last percent torque		1.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_init_data[6]	hours of flight		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_init_data[7]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d
engine_init_data[8]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d
engine_init_data[9]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.7. - ENGINE STATUS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
engine_stat_data[0]	minutes of flight		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[1]	old minutes of flight		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[2]	engine status		1	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[3]	starting engine		1	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[4]	number of engines		2	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[5]	engine is damaged		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[6]	transmission is damaged		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_stat_data[7]	NOT USED		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d
engine_stat_data[8]	NOT USED		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d
engine_stat_data[9]	NOT USED		0	int	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.8. - KINEMATICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kinemat_data[0]	GRAV_CONSTANT;	m/sec**2	9.810	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rwa_kine.d
kinemat_data[1]	SIN_AOA_LIMIT;	deg	0.642787610	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]veh_spec_kinematics_init	simnet/data/rwa_kine.d
kinemat_data[2]	COS_AOA_LIMIT;	deg	0.766044443	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]veh_spec_kinematics_init	simnet/data/rwa_kine.d
kinemat_data[3]	SIN_YAW_LIMIT;	deg	0.642787610	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rwa_kine.d
kinemat_data[4]	COS_YAW_LIMIT;	deg	0.766044443	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rwa_kine.d
kinemat_data[5]	DISPLAY_SPEED_LIMIT;		5.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rwa_kine.d
kinemat_data[6]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[7]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[8]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[9]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[10]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[11]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[12]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[13]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[14]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[15]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[16]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[17]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d
kinemat_data[18]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d

TABLE 5.1.8. - KINEMATICS DATA ARRAY [Continued]

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kinemat_data[19]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rwa_kine.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.9. - KINEMATICS INITIALIZATION DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kinemat_init_data[0]	positive unit velocity in X axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rw_ki_in.d
kinemat_init_data[1]	positive unit velocity in Y axis		1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]veh_spec_kinematics_init	simnet/data/rw_ki_in.d
kinemat_init_data[2]	positive unit velocity in Z axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]veh_spec_kinematics_init	simnet/data/rw_ki_in.d
kinemat_init_data[3]	negative unit velocity in X axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rw_ki_in.d
kinemat_init_data[4]	negative unit velocity in Y axis		-1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rw_ki_in.d
kinemat_init_data[5]	negative unit velocity in Z axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rw_ki_in.d
kinemat_init_data[6]	sine angle of attack		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[7]	cosine angle of attack		1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[8]	sine yaw		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[9]	cosine yaw		1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[10]	altitude		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[11]	body pitch		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[12]	body pitch offset		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[13]	velocity pitch		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d

TABLE 5.1.9. - KINEMATICS INITIALIZATION DATA ARRAY
[Continued]

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kinemat_init_data[14]	roll		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[15]	heading		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init	[rwa_kinemat.c]engine_simul	simnet/data/rw_ki_in.d
kinemat_init_data[16]	true airspeed		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[17]	indicated airspeed		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[18]	"g" force		1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[19]	vertical speed		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[20]	gravity component in X axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[21]	gravity component in Y axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[22]	gravity component in Z axis		-1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[23]	normal velocity component in X axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[24]	normal velocity component in Y axis		1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[25]	normal velocity component in Z axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[26]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[27]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[28]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d
kinemat_init_data[29]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_init		simnet/data/rw_ki_in.d

NOTE 1 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.10 - HELLFIRE MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
hellfr_miss_char[0]	HELLFIRE_ARM_TIME; hellfire missile arm time delay before firing in ticks [1.3 seconds]	ticks	20.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[1]	HELLFIRE_BURNOUT_TIME; time of powered flight for hellfire missile in ticks [2.4 seconds]	ticks	36.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init	simnet/data/ms_hf_ch.d
hellfr_miss_char[2]	HELLFIRE_MAX_FLIGHT_TIME; maximum flight time for the hellfire missile assumed in ticks [36.0 seconds]	ticks	540.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init	simnet/data/ms_hf_ch.d
hellfr_miss_char[3]	SPEED_0;		30.95953043	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[4]	THETA_0;		0.046542113	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[5]	SIN_UNGUIDE; sine of the delta pitch angle [4.0 degrees] for an unguided hellfire missile		0.069756474	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[6]	COS_UNGUIDE; cosine of the delta pitch angle [4.0 degrees] for an unguided hellfire missile		0.997564050	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[7]	SIN_CLIMB; sine of the delta pitch angle [3.5 degrees] for a climbing hellfire missile		0.001072424	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[8]	COS_CLIMB; cosine of the delta pitch angle [3.5 degrees] for a climbing hellfire missile		0.999991708	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[9]	SIN_LOCK; sine of the lock cone angle [9.0 degrees] for a locked-on hellfire missile		0.156434465	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[10]	COS_LOCK; cosine of the lock cone angle [9.0 degrees] for a locked-on hellfire missile		0.987688341	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[11]	COS_TERM; cosine of the terminal pitch angle [76.0 degrees] for a locked-on hellfire missile		0.241921896	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[12]	COS_LOSE; cosine of the pitch angle [20.0 degrees] for a loss-of-lock-on hellfire missile		0.939692621	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[13]	NOT USED		0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d
hellfr_miss_char[14]	NOT USED		0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	simnet/data/ms_hf_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.11. - HELLFIRE MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
hellfire_miss_poly_deg[0]	HELLFIRE_TOF_DEG; polynomial degree for hellfire missile time-of-flight coefficient data array		default: 4 range: 0 to 9	int	default declaration miss_hellfr.c; [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_miss_poly_deg[1]	HELLFIRE_BURN_SPEED_DEG; polynomial degree for hellfire missile burn speed coefficient data array		default: 3 range: 0 to 9	int	default declaration miss_hellfr.c; [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_miss_poly_deg[2]	HELLFIRE_COAST_SPEED_DEG; polynomial degree for hellfire missile coast speed coefficient data array		default: 5 range: 0 to 9	int	default declaration miss_hellfr.c; [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.12. - HELLFIRE MISSILE TIME-OF-FLIGHT COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
hellfire_tof_coeff[0]	hellfire missile time-of-flight coefficient a_0; default to 1.2 seconds	ticks	18.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[1]	hellfire missile time-of-flight coefficient a_1	ticks/meter	3.1461816e-2	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[2]	hellfire missile time-of-flight coefficient a_2	ticks/m**2	3.1921274e-6	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[3]	hellfire missile time-of-flight coefficient a_3	ticks/m**3	3.5260413e-10	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[4]	hellfire missile time-of-flight coefficient a_4	ticks/m**4	-2.8469594e-14	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[5]	hellfire missile time-of-flight coefficient a_5	ticks/m**5	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[6]	hellfire missile time-of-flight coefficient a_6	ticks/m**6	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[7]	hellfire missile time-of-flight coefficient a_7	ticks/m**7	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[8]	hellfire missile time-of-flight coefficient a_8	ticks/m**8	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	simnet/data/ms_hf_tf.d
hellfire_tof_coeff[9]	hellfire missile time-of-flight coefficient a_9	ticks/m**9	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_calc_tof	

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.13. - HELLFIRE MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
hellfire_burn_speed_coeff[0]	hellfire missile burn speed coefficient a_0	meters	2.0014395e-2	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[1]	hellfire missile burn speed coefficient a_1	m/tick	6.7384206e-1	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[2]	hellfire missile burn speed coefficient a_2	m/tick**2	9.8007701e-3	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[3]	hellfire missile burn speed coefficient a_3	m/tick**3	-1.6782227e-4	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[4]	hellfire missile burn speed coefficient a_4	m/tick**4	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[5]	hellfire missile burn speed coefficient a_5	m/tick**5	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[6]	hellfire missile burn speed coefficient a_6	m/tick**6	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[7]	hellfire missile burn speed coefficient a_7	m/tick**7	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[8]	hellfire missile burn speed coefficient a_8	m/tick**8	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d
hellfire_burn_speed_coeff[9]	hellfire missile burn speed coefficient a_9	m/tick**9	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fire; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.14. - HELLFIRE MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
hellfire_coast_speed_coeff[0]	hellfire missile coast speed coefficient a_0	meters	4.2738447e+1	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[1]	hellfire missile coast speed coefficient a_1	m/tick	-4.1018613e-1	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[2]	hellfire missile coast speed coefficient a_2	m/tick**2	2.6023601e-3	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[3]	hellfire missile coast speed coefficient a_3	m/tick**3	-8.4870117e-6	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[4]	hellfire missile coast speed coefficient a_4	m/tick**4	1.3322932e-8	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[5]	hellfire missile coast speed coefficient a_5	m/tick**5	-7.9542005e-12	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[6]	hellfire missile coast speed coefficient a_6	m/tick**6	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[7]	hellfire missile coast speed coefficient a_7	m/tick**7	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[8]	hellfire missile coast speed coefficient a_8	m/tick**8	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d
hellfire_coast_speed_coeff[9]	hellfire missile coast speed coefficient a_9	m/tick**9	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_fly	simnet/data/ms_hf_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.15 - MAVERICK MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
maverick_miss_char[0]	MAVERICK_ARM_TIME; maverick missile arm time delay before firing in ticks [1.3 seconds]	ticks	20.0	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[1]	MAVERICK_BURNOUT_TIME; time of powered flight for maverick missile in ticks [1.5 seconds]	ticks	22.5	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[2]	MAVERICK_MAX_FLIGHT_TIME; maximum flight time for the maverick missile assumed in ticks [60.0 seconds]	ticks	900.0	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init	simnet/data/ms_mk_ch.d
maverick_miss_char[3]	MAVERICK_LOCK_THRESHOLD; cosine squared of the lock threshold angle for the maverick missile [6.0 degrees]		0.989073800	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_detectibility	simnet/data/ms_mk_ch.d
maverick_miss_char[4]	MAVERICK_HOLD_THRESHOLD; cosine squared of the hold threshold angle for the maverick missile [10.0 degrees]		0.969846310	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_detectibility	simnet/data/ms_mk_ch.d
maverick_miss_char[5]	SPEED_0;		28.33333333	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[6]	THETA_0;		0.046542113	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[7]	SIN_UNGUIDE; sine of level flight [0.0 degrees pitch] for an unguided maverick missile		0.0	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[8]	COS_UNGUIDE; cosine of level flight [0.0 degrees pitch] for an unguided maverick missile		1.0	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[9]	SIN_CLIMB; sine of the delta pitch angle [3.5 degrees] for a climbing maverick missile		0.004072424	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[10]	COS_CLIMB; cosine of the delta pitch angle [3.5 degrees] for a climbing maverick missile		0.999991708	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[11]	SIN_LOCK; sine of the lock cone angle [5.0 degrees] for a locked-on maverick missile		0.087155743	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[12]	COS_LOCK; cosine of the lock cone angle [5.0 degrees] for a locked-on maverick missile		0.996194698	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[13]	COS_TERM; cosine of the terminal angle [80.0 degrees] for a locked-on maverick missile		0.173648178	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d
maverick_miss_char[14]	COS_LOSE; cosine of the angle [20.0 degrees] for a loss-of-lock-on maverick missile		0.939692621	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.16. - MAVERICK MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
maverick_miss_poly_deg[0]	MAVERICK_BURN_SPEED_DEG; polynomial degree for maverick missile burn speed coefficient data array		default: 1 range: 0 to 9	int	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fire; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_bs.d
maverick_miss_poly_deg[1]	MAVERICK_COAST_SPEED_DEG; polynomial degree for maverick missile coast speed coefficient data array		default: 3 range: 0 to 9	int	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_cs.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.17. - MAVERICK MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
maverick_burn_speed_coeff[0]	maverick missile burn speed coefficient a_0; default is 67.0 m/sec	m/tick	0.03333333	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fire; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_bs.d
maverick_burn_speed_coeff[1]	maverick missile burn speed coefficient a_1; default is 274.9732662 m/sec**2	m/tick**2	1.25777777	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fire; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_bs.d
maverick_burn_speed_coeff[2]	maverick missile burn speed coefficient a_2	m/tick**3	0.0	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fire; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_bs.d
maverick_burn_speed_coeff[3]	maverick missile burn speed coefficient a_3	m/tick**4	0.0	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fire; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_bs.d
maverick_burn_speed_coeff[4]	maverick missile burn speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fire; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.18. - MAVERICK MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
maverick_coast_speed_coeff[0]	maverick missile coast speed coefficient a_0; default is 327.2858074 m/sec	m/tick	30.46972849	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[1]	maverick missile coast speed coefficient a_1; default is -21.4609544 m/sec**2	m/tick**2	-9.7721160e-2	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[2]	maverick missile coast speed coefficient a_2; default is 0.8227650 m/sec**3	m/tick**3	1.2433925e-4	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[3]	maverick missile coast speed coefficient a_3; default is -0.0133200 m/sec**4	m/tick**4	-5.4061501e-8	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[4]	maverick missile coast speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_init	[miss_maverck.c]missile_maverick_init; [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.19 - STINGER MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
stinger_miss_char[0]	STINGER_BURNOUT_TIME; time of powered flight for stinger missile in ticks [1.275 seconds]	ticks	19.125	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[1]	STINGER_MAX_FLIGHT_TIME; maximum flight time for the stinger missile assumed in ticks [26.667 seconds]	ticks	400.000	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init	simnet/data/ms_st_ch.d
stinger_miss_char[2]	STINGER_LOCK_THRESHOLD; cosine squared of the lock threshold angle for the stinger missile [12.5 degrees]		0.953153895	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_pre_launch; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[3]	SPEED_0; default is 800.0 m/sec	m/tick	53.33333333	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[4]	THETA_0; default is 15.0 deg/sec	rad/tick	0.0174	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[5]	INVEST_DIST_SQ; default distance is 300 m	m**2	90000.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[6]	FUZE_DIST_SQ; default distance is 20 m	m**2	400.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[7]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[8]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[9]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[10]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[11]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[12]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[13]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d
stinger_miss_char[14]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.20. - STINGER MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
stinger_miss_poly_deg[0]	polynomial degree for stinger missile burn speed coefficient data array		1	int	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init	simnet/data/ms_st_bs.d
stinger_miss_poly_deg[1]	polynomial degree for stinger missile coast speed coefficient data array		3	int	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init	simnet/data/ms_st_cs.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.21. - STINGER MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
stinger_burn_speed_coeff[0]	stinger missile burn speed coefficient a_0	m/tick	1.9	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_bs.d
stinger_burn_speed_coeff[1]	stinger missile burn speed coefficient a_1	m/tick**2	2.689324619	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.22. - STINGER MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
stinger_coast_speed_coeff[0]	stinger missile coast speed coefficient a_0	m/tick	56.73662833	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_cs.d
stinger_coast_speed_coeff[1]	stinger missile coast speed coefficient a_1	m/tick**2	-0.182369351	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_cs.d
stinger_coast_speed_coeff[2]	stinger missile coast speed coefficient a_2	m/tick**3	2.3302001e-4	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_cs.d
stinger_coast_speed_coeff[3]	stinger missile coast speed coefficient a_3	m/tick**4	-1.0176282e-7	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.23 - TOW MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_miss_char[0]	TOW_BURNOUT_TIME; time of powered flight for tow missile in ticks [1.6 seconds]	ticks	24.0	REAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ch.d
tow_miss_char[1]	TOW_RANGE_LIMIT_TIME; range limit time for the tow missile in ticks [17.89 seconds]; at this point the wire is cut, but the missile is allowed to fly to the maximum flight time	ticks	268.35	REAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ch.d
tow_miss_char[2]	TOW_MAX_FLIGHT_TIME; maximum flight time for the tow missile in ticks; cosine of the max turn is greater than 1.0 beyond this point	ticks	300.00	REAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_ch.d
tow_miss_char[3]	NOT USED		0.0	REAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_ch.d
tow_miss_char[4]	NOT USED		0.0	REAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.24. - TOW MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_miss_poly_deg[0]	polynomial degree for tow missile burn speed coefficient data array		2	int	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_bs.d
tow_miss_poly_deg[1]	polynomial degree for tow missile coast speed coefficient data array		3	int	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_cs.d
tow_miss_poly_deg[2]	polynomial degree for each tow missile burn turn coefficient data sub-array of the tow missile burn turn coefficient data array structure		1	int	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_bt.d
tow_miss_poly_deg[3]	polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array structure		3	int	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_ct.d
tow_miss_poly_deg[4]	NOT USED		0	int	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init	simnet/data/ms_tw_ct.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.25. - TOW MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_burn_speed_coeff[0]	tow missile burn speed coefficient a_0; default value is 67.0 m/sec	m/tick	4.466666667	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bs.d
tow_burn_speed_coeff[1]	tow missile burn speed coefficient a_1; default value is 274.9732662 m/sec**2	m/tick**2	1.222103405	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bs.d
tow_burn_speed_coeff[2]	tow missile burn speed coefficient a_2; default value is -82.7057910 m/sec**3	m/tick**3	-0.024532086	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bs.d
tow_burn_speed_coeff[3]	tow missile burn speed coefficient a_3	m/tick**4	0.0	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bs.d
tow_burn_speed_coeff[4]	tow missile burn speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.26. - TOW MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_coast_speed_coeff[0]	tow missile coast speed coefficient a_0; default value is 327.2858074 m/sec	m/tick	21.81905383	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_speed_coeff[1]	tow missile coast speed coefficient a_1; default value is -21.4609544 m/sec**2	m/tick**2	-9.5382019e-2	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_speed_coeff[2]	tow missile coast speed coefficient a_2; default value is 0.8227650 m/sec**3	m/tick**3	2.4378222e-4	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_speed_coeff[3]	tow missile coast speed coefficient a_3; default value is -0.0133200 m/sec**4	m/tick**4	-2.6311111e-7	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_speed_coeff[4]	tow missile coast speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.27. - TOW MISSILE BURN TURN COEFFICIENT DATA STRUCTURE

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_burn_turn_coeff.deg	polynomial degree for each tow missile burn turn coefficient data sub-array of the tow missile burn turn coefficient data array structure		1	int	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d
tow_burn_turn_coeff.side_coeff[0]	tow missile cosine of maximum side turn during burn coefficient a_0	cos(rad)/tick	0.999976868652	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d
tow_burn_turn_coeff.side_coeff[1]	tow missile cosine of maximum side turn during burn coefficient a_1	cos(rad)/tick**2	-3.5933955e-7	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d
tow_burn_turn_coeff.up_coeff[0]	tow missile cosine of maximum up turn during burn coefficient a_0	cos(rad)/tick*	0.999960667258	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d
tow_burn_turn_coeff.up_coeff[1]	tow missile cosine of maximum up turn during burn coefficient a_1	cos(rad)/tick**2	-3.1492328e-6	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d
tow_burn_turn_coeff.down_coeff[0]	tow missile cosine of maximum down turn during burn coefficient a_0	cos(rad)/tick	0.999978909989	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d
tow_burn_turn_coeff.down_coeff[1]	tow missile cosine of maximum down turn during burn coefficient a_1	cos(rad)/tick**2	-7.8194991e-9	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_bt.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.
int is a "C" type for integer.

TABLE 5.1.28. - TOW MISSILE COAST TURN COEFFICIENT DATA STRUCTURE

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_coast_turn_coeff.deg	polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array structure		3	int	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.side_coeff[0]	tow missile cosine of maximum side turn during coast coefficient a_0	cos(rad)/tick	0.99995112518	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.side_coeff[1]	tow missile cosine of maximum side turn during coast coefficient a_1	cos(rad)/tick**2	8.96333e-7	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.side_coeff[2]	tow missile cosine of maximum side turn during coast coefficient a_2	cos(rad)/tick**3	-5.995375e-9	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.side_coeff[3]	tow missile cosine of maximum side turn during coast coefficient a_3	cos(rad)/tick**4	1.162225e-11	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.up_coeff[0]	tow missile cosine of maximum up turn during coast coefficient a_0	cos(rad)/tick	0.9998498495	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.up_coeff[1]	tow missile cosine of maximum up turn during coast coefficient a_1	cos(rad)/tick**2	1.657779e-6	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.up_coeff[2]	tow missile cosine of maximum up turn during coast coefficient a_2	cos(rad)/tick**3	-8.231861e-9	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.up_coeff[3]	tow missile cosine of maximum up turn during coast coefficient a_3	cos(rad)/tick**4	1.381832e-11	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.down_coeff[0]	tow missile cosine of maximum down turn during coast coefficient a_0	cos(rad)/tick	0.9999714014	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.down_coeff[1]	tow missile cosine of maximum down turn during coast coefficient a_1	cos(rad)/tick**2	3.382077e-7	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.down_coeff[2]	tow missile cosine of maximum down turn during coast coefficient a_2	cos(rad)/tick**3	-1.601259e-9	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.down_coeff[3]	tow missile cosine of maximum down turn during coast coefficient a_3	cos(rad)/tick**4	2.623014e-12	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.
int is a "C" type for integer.

TABLE 5.1.29 - ADAT MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_miss_char[0]	ADAT_BURNOUT_TIME; time of powered flight for adat missile in ticks [3.2 seconds]	ticks	48.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ch.d
adat_miss_char[1]	ADAT_MAX_FLIGHT_TIME; maximum flight time for the adat missile in ticks [20.0 seconds]	ticks	300.00	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ch.d
adat_miss_char[2]	INVEST_DIST_SQ; default value is 300 meters squared	m**2	90000.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ch.d
adat_miss_char[3]	HELO_FUZE_DIST_SQ; default value is 7 meters squared	m**2	49.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire	simnet/data/ms_ad_ch.d
adat_miss_char[4]	AIR_FUZE_DIST_SQ; default value is 14 meters squared	m**2	196.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire	simnet/data/ms_ad_ch.d
adat_miss_char[5]	ADAT_TEMP_BIAS_TIME; time of temporal bias for adat missile in ticks [4.0 seconds]	ticks	60.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ch.d
adat_miss_char[6]	CLOSE_RANGE;	m	2200.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire	simnet/data/ms_ad_ch.d
adat_miss_char[7]	NOT USED		0.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ch.d
adat_miss_char[8]	NOT USED		0.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ch.d
adat_miss_char[9]	NOT USED		0.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.30. - ADAT MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_miss_poly_deg[0]	polynomial degree for adat missile burn speed coefficient data array		2	int	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_bs.d
adat_miss_poly_deg[1]	polynomial degree for adat missile coast speed coefficient data array		4	int	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_cs.d
adat_miss_poly_deg[2]	polynomial degree for cosine of adat missile maximum turn during burn coefficient data array		3	int	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_bt.d
adat_miss_poly_deg[3]	polynomial degree for cosine of adat missile maximum turn during coast coefficient data array		5	int	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ct.d
adat_miss_poly_deg[4]	polynomial degree for adat missile temporal bias coefficient data array		4	int	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_tb.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.31. - ADAT MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_burn_speed_coeff[0]	adat missile burn speed coefficient a_0	m/tick	2.296	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[1]	adat missile burn speed coefficient a_1	m/tick**2	0.72990856	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[2]	adat missile burn speed coefficient a_2	m/tick**3	0.013310932	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[3]	adat missile burn speed coefficient a_3	m/tick**4	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[4]	adat missile burn speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[5]	adat missile burn speed coefficient a_5	m/tick**6	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[6]	adat missile burn speed coefficient a_6	m/tick**7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[7]	adat missile burn speed coefficient a_7	m/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[8]	adat missile burn speed coefficient a_8	m/tick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[9]	adat missile burn speed coefficient a_9	m/tick**10	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.32. - ADAT MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_coast_speed_coeff[0]	adat missile coast speed coefficient a_0	m/tick	105.52162	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[1]	adat missile coast speed coefficient a_1	m/tick**2	-1.0157285	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[2]	adat missile coast speed coefficient a_2	m/tick**3	5.6124330e-3	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[3]	adat missile coast speed coefficient a_3	m/tick**4	-1.6262608e-5	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[4]	adat missile coast speed coefficient a_4	m/tick**5	1.8991982e-8	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[5]	adat missile coast speed coefficient a_5	m/tick**6	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[6]	adat missile coast speed coefficient a_6	m/tick**7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[7]	adat missile coast speed coefficient a_7	m/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[8]	adat missile coast speed coefficient a_8	m/tick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d
adat_coast_speed_coeff[9]	adat missile coast speed coefficient a_9	m/tick**10	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.33. - ADAT MISSILE BURN TURN COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR - CALCULATED	CSU WHERE USED	DATA SOURCE
adat_burn_turn_coeff[0]	adat missile cosine of maximum turn during burn coefficient a_0	cos(rad)/tick	0.999993	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[1]	adat missile cosine of maximum turn during burn coefficient a_1	cos(rad)/tick**2	-6.2386917e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[2]	adat missile cosine of maximum turn during burn coefficient a_2	cos(rad)/tick**3	1.6146426e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[3]	adat missile cosine of maximum turn during burn coefficient a_3	cos(rad)/tick**4	-9.720142e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[4]	adat missile cosine of maximum turn during burn coefficient a_4	cos(rad)/tick**5	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[5]	adat missile cosine of maximum turn during burn coefficient a_5	cos(rad)/tick**6	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[6]	adat missile cosine of maximum turn during burn coefficient a_6	cos(rad)/tick**7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[7]	adat missile cosine of maximum turn during burn coefficient a_7	cos(rad)/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[8]	adat missile cosine of maximum turn during burn coefficient a_8	cos(rad)/tick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adat_burn_turn_coeff[9]	adat missile cosine of maximum turn during burn coefficient a_9	cos(rad)/tick**10	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.34. - ADAT MISSILE COAST TURN COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_coast_turn_coeff[0]	adat missile cosine of maximum turn during coast coefficient a_0	cos(rad)/tick	0.99753111	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[1]	adat missile cosine of maximum turn during coast coefficient a_1	cos(rad)/tick**2	5.5817986e-5	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[2]	adat missile cosine of maximum turn during coast coefficient a_2	cos(rad)/tick**3	-5.1276276e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[3]	adat missile cosine of maximum turn during coast coefficient a_3	cos(rad)/tick**4	2.2388593e-9	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[4]	adat missile cosine of maximum turn during coast coefficient a_4	cos(rad)/tick**5	-5.1964622e-12	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[5]	adat missile cosine of maximum turn during coast coefficient a_5	cos(rad)/tick**6	4.5499104e-15	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[6]	adat missile cosine of maximum turn during coast coefficient a_6	cos(rad)/tick**7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[7]	adat missile cosine of maximum turn during coast coefficient a_7	cos(rad)/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[8]	adat missile cosine of maximum turn during coast coefficient a_8	cos(rad)/tick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[9]	adat missile cosine of maximum turn during coast coefficient a_9	cos(rad)/tick**10	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.35. - ADAT MISSILE TEMPORAL BIAS COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_temp_bias_coeff[0]	adat missile temporal bias coefficient a_0	cos(rad)/tick	5.3105657e-2	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[1]	adat missile temporal bias coefficient a_1	cos(rad)/tick**2	7.1795817e-2	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[2]	adat missile temporal bias coefficient a_2	cos(rad)/tick**3	1.8084646e-2	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[3]	adat missile temporal bias coefficient a_3	cos(rad)/tick**4	-6.0083762e-4	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[4]	adat missile temporal bias coefficient a_4	cos(rad)/tick**5	4.6761091e-6	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[5]	adat missile temporal bias coefficient a_5	cos(rad)/tick**6	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[6]	adat missile temporal bias coefficient a_6	cos(rad)/tick**7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[7]	adat missile temporal bias coefficient a_7	cos(rad)/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[8]	adat missile temporal bias coefficient a_8	cos(rad)/tick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_temp_bias_coeff[9]	adat missile temporal bias coefficient a_9	cos(rad)/tick**10	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.36 - ATGM MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_miss_char[0]	TOW_BURNOUT_TIME; time of powered flight for tow missile in ticks [1.6 seconds]	ticks	24.0	REAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ch.d
tow_miss_char[1]	TOW_RANGE_LIMIT_TIME; range limit time for the tow missile in ticks [17.89 seconds]; at this point the wire is cut, but the missile is allowed to fly to the maximum flight time	ticks	268.35	REAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ch.d
tow_miss_char[2]	TOW_MAX_FLIGHT_TIME; maximum flight time for the tow missile in ticks; cosine of the max turn is greater than 1.0 beyond this point	ticks	200.00	REAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_ch.d
tow_miss_char[3]	ATGM_TURN_FACTOR; ATGM turn factor for wider turning capability with respect to TOW		0.9	REAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_ch.d
tow_miss_char[4]	NOT USED		0.0	REAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.37. - ATGM MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_miss_poly_deg[0]	polynomial degree for tow missile burn speed coefficient data array		2	int	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_bs.d
tow_miss_poly_deg[1]	polynomial degree for tow missile coast speed coefficient data array		3	int	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_cs.d
tow_miss_poly_deg[2]	polynomial degree for each tow missile burn turn coefficient data sub-array of the tow missile burn turn coefficient data array structure		1	int	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_bt.d
tow_miss_poly_deg[3]	polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array structure		3	int	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_ct.d
tow_miss_poly_deg[4]	NOT USED		0	int	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_ct.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.38. - ATGM MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_burn_speed_coeff[0]	tow missile burn speed coefficient a_0; default value is 67.0 m/sec	m/tick	4.466666667	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fire; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bs.d
tow_burn_speed_coeff[1]	tow missile burn speed coefficient a_1; default value is 274.9732662 m/sec**2	m/tick**2	1.222103405	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fire; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bs.d
tow_burn_speed_coeff[2]	tow missile burn speed coefficient a_2; default value is -82.7057910 m/sec**3	m/tick**3	-0.024532086	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fire; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bs.d
tow_burn_speed_coeff[3]	tow missile burn speed coefficient a_3	m/tick**4	0.0	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fire; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bs.d
tow_burn_speed_coeff[4]	tow missile burn speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fire; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.39. - ATGM MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_coast_speed_coeff[0]	tow missile coast speed coefficient a_0; default value is 327.2858074 m/sec	m/tick	21.81905383	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_coast_speed_coeff[1]	tow missile coast speed coefficient a_1; default value is -21.4609544 m/sec**2	m/tick**2	-9.5382019e-2	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_coast_speed_coeff[2]	tow missile coast speed coefficient a_2; default value is 0.8227650 m/sec**3	m/tick**3	2.4378222e-4	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_coast_speed_coeff[3]	tow missile coast speed coefficient a_3; default value is -0.0133200 m/sec**4	m/tick**4	-2.6311111e-7	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_coast_speed_coeff[4]	tow missile coast speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.40. - ATGM MISSILE BURN TURN COEFFICIENT DATA STRUCTURE

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR ~CALCULATED	CSU WHERE USED	DATA SOURCE
tow_burn_turn_coeff.deg	polynomial degree for each tow missile burn turn coefficient data sub-array of the tow missile burn turn coefficient data array structure		1	int	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.side_coeff[0]	tow missile cosine of maximum side turn during burn coefficient a_0	cos(rad)/tick	0.999976868652	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.side_coeff[1]	tow missile cosine of maximum side turn during burn coefficient a_1	cos(rad)/tick**2	-3.5933955e-7	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.up_coeff[0]	tow missile cosine of maximum up turn during burn coefficient a_0	cos(rad)/tick*	0.999960667258	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.up_coeff[1]	tow missile cosine of maximum up turn during burn coefficient a_1	cos(rad)/tick**2	-3.1492328e-6	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.down_coeff[0]	tow missile cosine of maximum down turn during burn coefficient a_0	cos(rad)/tick	0.999978909989	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.down_coeff[1]	tow missile cosine of maximum down turn during burn coefficient a_1	cos(rad)/tick**2	-7.8194991e-9	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_bt.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.
int is a "C" type for integer.

TABLE 5.1.41. - ATGM MISSILE COAST TURN COEFFICIENT DATA STRUCTURE

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_coast_turn_coeff.deg	polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array structure		3	int	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.side_coeff[0]	tow missile cosine of maximum side turn during coast coefficient a_0	cos(rad)/tick	0.99995112518	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.side_coeff[1]	tow missile cosine of maximum side turn during coast coefficient a_1	cos(rad)/tick**2	8.96333e-7	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.side_coeff[2]	tow missile cosine of maximum side turn during coast coefficient a_2	cos(rad)/tick**3	-5.995375e-9	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.side_coeff[3]	tow missile cosine of maximum side turn during coast coefficient a_3	cos(rad)/tick**4	1.162225e-11	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.up_coeff[0]	tow missile cosine of maximum up turn during coast coefficient a_0	cos(rad)/tick	0.9998498495	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.up_coeff[1]	tow missile cosine of maximum up turn during coast coefficient a_1	cos(rad)/tick**2	1.657779e-6	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.up_coeff[2]	tow missile cosine of maximum up turn during coast coefficient a_2	cos(rad)/tick**3	-8.231861e-9	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.up_coeff[3]	tow missile cosine of maximum up turn during coast coefficient a_3	cos(rad)/tick**4	1.381832e-11	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.down_coeff[0]	tow missile cosine of maximum down turn during coast coefficient a_0	cos(rad)/tick	0.9999714014	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.down_coeff[1]	tow missile cosine of maximum down turn during coast coefficient a_1	cos(rad)/tick**2	3.382077e-7	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.down_coeff[2]	tow missile cosine of maximum down turn during coast coefficient a_2	cos(rad)/tick**3	-1.601259e-9	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_coast_turn_coeff.down_coeff[3]	tow missile cosine of maximum down turn during coast coefficient a_3	cos(rad)/tick**4	2.623014e-12	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.
int is a "C" type for integer.

TABLE 5.1.42 - KEM MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_miss_char[0]	KEM_BURNOUT_TIME; time of powered flight for kem missile in ticks [3.2 seconds]	ticks	48.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ch.d
kem_miss_char[1]	KEM_MAX_FLIGHT_TIME; maximum flight time for the kem missile in ticks [20.0 seconds]	ticks	300.00	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ch.d
kem_miss_char[2]	KEM_TO_MACH5_FACTOR; speed factor to raise from ADAT to KEM; just after burnout, the ADAT has a maximum velocity of 230 m/sec, while the KEM has a maximum velocity of 1524 m/sec		6.626	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ch.d
kem_miss_char[3]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire	simnet/data/ms_km_ch.d
kem_miss_char[4]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire	simnet/data/ms_km_ch.d
kem_miss_char[5]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ch.d
kem_miss_char[6]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire	simnet/data/ms_km_ch.d
kem_miss_char[7]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_ch.d
kem_miss_char[8]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_ch.d
kem_miss_char[9]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.43. - KEM MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_miss_poly_deg[0]	polynomial degree for kem missile burn speed coefficient data array		2	int	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_bs.d
kem_miss_poly_deg[1]	polynomial degree for kem missile coast speed coefficient data array		4	int	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_cs.d
kem_miss_poly_deg[2]	polynomial degree for cosine of kem missile maximum turn during burn coefficient data array		3	int	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_bt.d
kem_miss_poly_deg[3]	polynomial degree for cosine of kem missile maximum turn during coast coefficient data array		5	int	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_ct.d
kem_miss_poly_deg[4]	NOT USED		0	int	default declaration miss_kem.c		

NOTE 1 int is a "C" type for integer.

TABLE 5.1.44. - KEM MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_burn_speed_coeff[0]	kem missile burn speed coefficient a_0	m/tick	2.296	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[1]	kem missile burn speed coefficient a_1	m/tick**2	0.72990856	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[2]	kem missile burn speed coefficient a_2	m/tick**3	0.013310932	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[3]	kem missile burn speed coefficient a_3	m/tick**4	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[4]	kem missile burn speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[5]	kem missile burn speed coefficient a_5	m/tick**6	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[6]	kem missile burn speed coefficient a_6	m/tick**7	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[7]	kem missile burn speed coefficient a_7	m/tick**8	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[8]	kem missile burn speed coefficient a_8	m/tick**9	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d
kem_burn_speed_coeff[9]	kem missile burn speed coefficient a_9	m/tick**10	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.45. - KEM MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_coast_speed_coeff[0]	kem missile coast speed coefficient a_0	m/tick	105.52162	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[1]	kem missile coast speed coefficient a_1	m/tick**2	-1.0157285	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[2]	kem missile coast speed coefficient a_2	m/tick**3	5.6124330e-3	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[3]	kem missile coast speed coefficient a_3	m/tick**4	-1.6262608e-5	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[4]	kem missile coast speed coefficient a_4	m/tick**5	1.8991982e-8	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[5]	kem missile coast speed coefficient a_5	m/tick**6	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[6]	kem missile coast speed coefficient a_6	m/tick**7	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[7]	kem missile coast speed coefficient a_7	m/tick**8	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[8]	kem missile coast speed coefficient a_8	m/tick**9	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d
kem_coast_speed_coeff[9]	kem missile coast speed coefficient a_9	m/tick**10	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.46. - KEM MISSILE BURN TURN COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_burn_turn_coeff[0]	kem missile cosine of maximum turn during burn coefficient a_0	cos(rad)/tick	0.999993	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[1]	kem missile cosine of maximum turn during burn coefficient a_1	cos(rad)/tick**2	-6.2386917e-7	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[2]	kem missile cosine of maximum turn during burn coefficient a_2	cos(rad)/tick**3	1.6146426e-7	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[3]	kem missile cosine of maximum turn during burn coefficient a_3	cos(rad)/tick**4	-9.720142e-7	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[4]	kem missile cosine of maximum turn during burn coefficient a_4	cos(rad)/tick**5	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[5]	kem missile cosine of maximum turn during burn coefficient a_5	cos(rad)/tick**6	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[6]	kem missile cosine of maximum turn during burn coefficient a_6	cos(rad)/tick**7	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[7]	kem missile cosine of maximum turn during burn coefficient a_7	cos(rad)/tick**8	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[8]	kem missile cosine of maximum turn during burn coefficient a_8	cos(rad)/tick**9	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d
kem_burn_turn_coeff[9]	kem missile cosine of maximum turn during burn coefficient a_9	cos(rad)/tick**10	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_bt.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.47. - KEM MISSILE COAST TURN COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_coast_turn_coeff[0]	kem missile cosine of maximum turn during coast coefficient a_0	cos(rad)/tick	0.99753111	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[1]	kem missile cosine of maximum turn during coast coefficient a_1	cos(rad)/tick**2	5.5817986e-5	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[2]	kem missile cosine of maximum turn during coast coefficient a_2	cos(rad)/tick**3	-5.1276276e-7	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[3]	kem missile cosine of maximum turn during coast coefficient a_3	cos(rad)/tick**4	2.2388593e-9	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[4]	kem missile cosine of maximum turn during coast coefficient a_4	cos(rad)/tick**5	-5.1964622e-12	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[5]	kem missile cosine of maximum turn during coast coefficient a_5	cos(rad)/tick**6	4.5499104e-15	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[6]	kem missile cosine of maximum turn during coast coefficient a_6	cos(rad)/tick**7	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[7]	kem missile cosine of maximum turn during coast coefficient a_7	cos(rad)/tick**8	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[8]	kem missile cosine of maximum turn during coast coefficient a_8	cos(rad)/tick**9	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_turn_coeff[9]	kem missile cosine of maximum turn during coast coefficient a_9	cos(rad)/tick**10	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.48. - NLOS MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
nlos_miss_char[0]	NLOS_LOCK_THRESHOLD;		0.953153895	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[1]	NLOS_MAX_TURN_ANGLE;	radians/ticks	0.03490659	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[2]	NLOS_VERTICAL_FLIGHT_TIME;	ticks	48.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init	simnet/data/ms_nl_ch.d
nlos_miss_char[3]	NLOS_DECLINE_FLIGHT_TIME;	ticks	105.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_detectibility	simnet/data/ms_nl_ch.d
nlos_miss_char[4]	NLOS_LEVEL_FLIGHT_TIME;	ticks	140.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_detectibility	simnet/data/ms_nl_ch.d
nlos_miss_char[5]	NLOS_ARM_TIME; nlos missile arm time delay before firing in ticks [1.3 seconds]	ticks	20.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[6]	NLOS_BURNOUT_TIME; time of powered flight for nlos missile in ticks [1.5 seconds]	ticks	22.5	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[7]	NLOS_MAX_FLIGHT_TIME; maximum flight time for the nlos missile assumed in ticks [120.0 seconds]	ticks	8000.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[8]	SPEED_0;		11.33333333	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[9]	SPEED_1;		5.333333333	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[10]	THETA_0;		0.013962634	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[11]	SIN_UNGUIDE; sine of level flight [4.0 degrees pitch] for an unguided nlos missile		0.069756474	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[12]	COS_UNGUIDE; cosine of level flight [4.0 degrees pitch] for an unguided nlos missile		0.997564050	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[13]	SIN_CLIMB; sine of the delta pitch angle [3.5 degrees] for a climbing nlos missile		0.004072424	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[14]	COS_CLIMB; cosine of the delta pitch angle [3.5 degrees] for a climbing nlos missile		0.999991708	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[15]	SIN_LOCK; sine of the lock cone angle [9.0 degrees] for a locked-on nlos missile		0.156434465	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[16]	COS_LOCK; cosine of the lock cone angle [9.0 degrees] for a locked-on nlos missile		0.987688341	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[17]	COS_TERM; cosine of the terminal angle [0.0 degrees] for a locked-on nlos missile		0.984807753	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[18]	COS_LOSE; cosine of the angle [20.0 degrees] for a loss-of-lock-on nlos missile		0.939692621	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[19]	NOT USED		0.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.49. - NLOS MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
nlos_miss_poly_deg[0]	NLOS_BURN_SPEED_DEG; polynomial degree for nlos missile burn speed coefficient data array		default: 1 range: 0 to 9	int	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fire; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_bs.d
nlos_miss_poly_deg[1]	NLOS_COAST_SPEED_DEG; polynomial degree for nlos missile coast speed coefficient data array		default: 3 range: 0 to 9	int	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_cs.d
nlos_miss_poly_deg[2]			0	int	default declaration miss_nlos.c		
nlos_miss_poly_deg[3]			0	int	default declaration miss_nlos.c		
nlos_miss_poly_deg[4]			0	int	default declaration miss_nlos.c		

NOTE 1 int is a "C" type for integer.

TABLE 5.1.50. - NLOS MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOTE 1]	DEFAULT VALUE	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
nlos_burn_speed_coeff[0]	nlos missile burn speed coefficient a_0; default is 67.0 m/sec	m/tick	0.03333333	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fire; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_bs.d
nlos_burn_speed_coeff[1]	nlos missile burn speed coefficient a_1; default is 274.9732662 m/sec**2	m/tick**2	1.25777777	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fire; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_bs.d
nlos_burn_speed_coeff[2]	nlos missile burn speed coefficient a_2	m/tick**3	0.0	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fire; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_bs.d
nlos_burn_speed_coeff[3]	nlos missile burn speed coefficient a_3	m/tick**4	0.0	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fire; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_bs.d
nlos_burn_speed_coeff[4]	nlos missile burn speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fire; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.51. - NLOS MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
nlos_coast_speed_coeff[0]	nlos missile coast speed coefficient a_0; default is 327.2858074 m/sec	m/tick	30.46972849	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_cs.d
nlos_coast_speed_coeff[1]	nlos missile coast speed coefficient a_1; default is -21.4609544 m/sec**2	m/tick**2	-9.7721160e-2	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_cs.d
nlos_coast_speed_coeff[2]	nlos missile coast speed coefficient a_2; default is 0.8227650 m/sec**3	m/tick**3	1.2433925e-4	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_cs.d
nlos_coast_speed_coeff[3]	nlos missile coast speed coefficient a_3; default is -0.0133200 m/sec**4	m/tick**4	-5.4061501e-8	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_cs.d
nlos_coast_speed_coeff[4]	nlos missile coast speed coefficient a_4	m/tick**5	0.0	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_init; [miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_cs.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.52 - HYDRA ROCKET CONFIGURATION DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
hydra_rkt_char[0]	hydra launcher postiion, X	m	4.5	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init;	/simnet/data/rwa_hydr.d
hydra_rkt_char[1]	hydra launcher positon, Y	m	0.5	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init;	/simnet/data/rwa_hydr.d
hydra_rkt_char[2]	hydra launcher postion, Z	m	-2.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init	/simnet/data/rwa_hydr.d
hydra_rkt_char[3]	mils of Soviet articulation	mils	104.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init	/simnet/data/rwa_hydr.d
hydra_rkt_char[4]	degrees of hull negative pitch	deg	-5.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init; [rwa_hydra.c]hydra_set_pylon_articulation	/simnet/data/rwa_hydr.d
hydra_rkt_char[5]	degress of maximum articulation	deg	19.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init	/simnet/data/rwa_hydr.d
hydra_rkt_char[6]	degrees of minumum articulation	deg	-15.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_init	/simnet/data/rwa_hydr.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.53 - HYDRA ROCKET CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
rkt_hydra_char[0]	M151_BURST_SPREAD; twin bursts which are 3 meters apart	m	1.5	REAL	default declaration rkt_ydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_set_pylon_articulation	/simnet/data/rkt_hydr.d
rkt_hydra_char[1]	M261_BURST_HEIGHT; release submunitions 180 feet	m	54.864	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_fire	/simnet/data/rkt_hydr.d
rkt_hydra_char[2]	M261_BURST_RANGE; 0 meters in front of target	m	0.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/simnet/data/rkt_hydr.d
rkt_hydra_char[3]	M261_BURST_SPREAD; twin bursts are 13 meters apart	m	6.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/simnet/data/rkt_hydr.d
rkt_hydra_char[4]	M255_BURST_RANGE; release darts 150 meters in front of target	m	150.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/simnet/data/rkt_hydr.d
rkt_hydra_char[5]	M255_BURST_SPREAD; twin bursts are 35 meters apart	m	16.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/simnet/data/rkt_hydr.d
rkt_hydra_char[6]	FLECH_60_MAX_RANGE; darts fly a total of 750 meters	m	750.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/simnet/data/rkt_hydr.d
rkt_hydra_char[7]	hydra minimum range	m	50.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_set_pylon_articulation	/simnet/data/rkt_hydr.d
rkt_hydra_char[8]	hydra maximum range for Soviet S-5 57mm rocket	m	5000.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init;	/simnet/data/rkt_hydr.d
rkt_hydra_char[9]	hydra maximum range for M151	m	7000.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_fire; [rkt_hydra.c]missile_hydra_set_pylon_articulation	/simnet/data/rkt_hydr.d
rkt_hydra_char[10]	hydra maximum range for M261	m	7000.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_fire; [rkt_hydra.c]missile_hydra_set_pylon_articulation	/simnet/data/rkt_hydr.d
rkt_hydra_char[11]	hydra maximum range for M255	m	3200.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_fire; [rkt_hydra.c]missile_hydra_set_pylon_articulation	/simnet/data/rkt_hydr.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.54. - SUBMUNITIONS M73 CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
sub_M73_char[0]	75% of gravity - (75% * (9.8m/sec**2)/225 ticks**2)	m/s**2/tick**2	0.03266667	REAL	default declaration sub_m73.c; [sub_m73.c]missile_m73_init	[sub_m73.c]missile_m73_init; [sub_m73.c]missile_m73_drop	/simnet/data/sub_m73.d
sub_M73_char[1]	bobmletties fall with +/- 8.8 degrees angular displacement	deg	15.6	REAL	default declaration sub_m73.c; [sub_m73.c]missile_m73_init	[sub_m73.c]missile_m73_init; [sub_m73.c]missile_m73_get_impact	/simnet/data/sub_m73.d
sub_M73_char[2]	bobmletties fall with +/- 12.35 degrees angular displacement	deg	22.7	REAL	default declaration sub_m73.c; [sub_m73.c]missile_m73_init	[sub_m73.c]missile_m73_init; [sub_m73.c]missile_m73_get_impact	/simnet/data/sub_m73.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

TABLE 5.1.55. - SUBMUNITIONS FLECHETTE CHARACTERISTICS DATA

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
sub_flech_char[0]	maximum speed < 100	m**2	10000.0	REAL	default declaration sub_flech.c; [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/sub_flec.d
sub_flech_char[1]	flechettes fly in a cylinder with a radius of 17.5 meters and a length of 750 meters	m**2	306.253	REAL	default declaration sub_flech.c; [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/sub_flec.d
sub_flech_char[2]	FLECH_60_MAX_RANGE; darts fly a total of 750 meters	m	750.0	REAL	default declaration sub_flech.c; [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/sub_flec.d

NOTE 1 int is a "C" type for integer.

TABLE 5.1.56. - FLECHETTE SPEED DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (NOTE 1)	DEFAULT VALUE	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
flechette_speed_coef[0]	flechette speed coefficient a_0	m/tick	41.75	REAL	default declaration sub_flech.c [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/flec_spd.d
flechette_speed_coef[1]	flechette speed coefficient a_1	m/tick/m	-0.20397254	REAL	default declaration sub_flech.c [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/flec_spd.d
flechette_speed_coef[2]	flechette speed coefficient a_2	m/tick/m**2	0.00022724278	REAL	default declaration sub_flech.c [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/flec_spd.d
flechette_speed_coef[3]	flechette speed coefficient a_3	m/tick/m**3	-0.00000008633	REAL	default declaration sub_flech.c [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/flec_spd.d
flechette_speed_coef[4]	flechette speed coefficient a_4	m/tick/m**4	0.0	REAL	default declaration sub_flech.c [sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_init; [sub_flech.c]missile_flechette_fly	/simnet/data/flec_spd.d

NOTE 1 one tick is equal to one frame or 1/15th of a second
NOTE 2 REAL is a "C" macro DEFINE for type float.

6. Notes.

NONE.

Appendix A - RWA AirNet Call Tree Structure.

The following appendix contains information for convenience in document maintenance and understanding of the overall CSCI architecture. This call tree is not all inclusive, i.e., it only contains the calls from the top-level down to the CSU of interest in this document. Other CSCs and CSUs have been included in the Call Tree for clarity and reference.

RWA AIRNET CALL TREE STRUCTURE

	1st	2nd	3rd	4th	5th	6th	7th	8th	
main									rwa_main.c
	enter_gracefully								main.c
		sim_state_idle							main.c
			printf						
			sim_state						main.c
		print_veh_logo							rwa_main.c
			clear_screen						
			printf						
			project_name						makevers.c
			version_str						makevers.c
			date_str						makevers.c
			select						
	network_set_exercise_id								rwa_main.c
	init_activ								rwa_config.c
	rwa_config_process_vehicle_type_string								rwa_config.c
		leftwing_stores							ammo.c
		ammo_set_all_quantity_zero							ammo.c
			ammo_indicators_require_updating						ammo.c
		rightwing_stores							ammo.c
		turret_stores							
		bzero							
		strncmp							
	main_process_pars_arg								read_pars.c
		sprintf							
		fopen							
		perror							
		printf							
		exit							
		subsys_ded_id							read_pars.c
		subsys_db_id							read_pars.c
		subsys_overlay_id							read_pars.c
		fgets							
		strtok							
		strcmp							
		libmsg_pars_file							read_pars.c
		sscanf							
		eye_to_screen_distance							read_pars.c
		vconfig_file1							read_pars.c
		vconfig_file2							read_pars.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
	veh_map_file							read_pars.c
	ammo_map_file							read_pars.c
	sdamage_file							read_pars.c
	devices_file							read_pars.c
	calib_file							read_pars.c
	assoc_def_file							read_pars.c
	het_calib_file							read_pars.c
	thresh_file							read_pars.c
	asid_map_file							read_pars.c
	idle_filter_file							read_pars.c
	sim_filter_file							read_pars.c
	priority_list_file							read_pars.c
	register_file							read_pars.c
	subsystems							read_pars.c
	atoi							
	cig_set_number_subsystems							
	default_db_name							read_pars.c
	default_db_version							read_pars.c
	db_override							read_pars.c
	cig_use_database_override_named							
	ded_override							read_pars.c
	set_ded_name							
	overlay_file							read_pars.c
	waypoint_list							read_pars.c
	constants_file							read_pars.c
	fclose							
atoi								
set_request_receive_size								
set_request_send_size								
set_assymetric_on								
need_to_fill_initial_munitions								rwa_main.c
debug								rwa_main.c
printf								
use_static_debug								
ntework_dont_really_open_up_ethernet								
force_other								rwa_main.c
f_velocity								rwa_main.c
sscanf								
cig_not_using_graphics								

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
guise_override								rwa_main.c
guise_to_use								rwa_main.c
print_help								rwa_main.c
	printf							
	network_get_network_device							
exit								
head_eye_tracker_enable								
use_print_checkb								
use_intervisibility_server								rwa_main.c
Intervisibility Init Setup								
keyboard_really_use								rwa_keybrd.c
use_keyboard								rwa_keybrd.c
het_enable_laser_effects								
het_set_damage_dir								
het_set_laser_series								
set_my_if								
v_pkt_verbose_mode								
print_overruns								rwa_main.c
strcpy								
get_default_db_name								read_pars.c
	default_db_name							read_pars.c
get_default_db_version								read_pars.c
	default_db_version							read_pars.c
movedata								
initial_activation								rwa_main.c
rva_turn_debug_on								
sound_dont_use								rwa_sound.c
	dont_use_sound							rwa_sound.c
	printf							
network_set_network_device								
db_subsys								rwa_main.c
cig_use_database_override_named								
ded_subsys								rwa_main.c
set_ded_name								
terrain_verbose_mode_on								
tads_set_intervisibility								rwa_tads.c
	strien							
	printf							
	how_to_do_intervisibility							rwa_tads.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
	strcpy							
set_cig_dev								
set_cig_mask								
fprintf								
sim_state_startup								main.c
	sim_state							main.c
simulate_state_machine								main.c
	sim_state							main.c
	initial_bbd							main.c
	bbd_init							
	printf							
	dtad_init							
	mem_assign_shared_memory							
	ser_heartbeat_init							
	idc_init							
	sound_init							rwa_sound.c
		dont_use_sound						rwa_sound.c
		sounds						rwa_mem.c
		fifo_init						
		sound_reset						rwa_sound.c
			dont_use_sound					rwa_sound.c
			sounds					rwa_mem.c
			fifo_init					
			sound_error					rwa_sound.c
				fprintf				
				fflush				
			veh_sound_array					rwa_sound.c
	status_init							rwa_status.c
		status_preset						rwa_status.c
			idc_values					
		equipment_status						rwa_status.c
		status_out						
	timers_init							
	pots_init							rwa_pots.c
		fopen						
		printf						
		exit						
		pil_cyc_roll_I						rwa_pots.c
		pil_cyc_roll_c						rwa_pots.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
		pil_cyc_roll_r						rwa_pots.c
		fscanf						
		pots_check_three						
		strcmp						
		pil_cyc_pitch_d						rwa_pots.c
		pil_cyc_pitch_c						rwa_pots.c
		pil_cyc_pitch_r						rwa_pots.c
		pil_pedal_l						rwa_pots.c
		pil_pedal_c						rwa_pots.c
		pil_pedal_r						rwa_pots.c
		pil_coll_d						rwa_pots.c
		pil_coll_r						rwa_pots.c
		pots_check_two						
		cpg_trav_l						rwa_pots.c
		cpg_trav_c						rwa_pots.c
		cpg_trav_r						rwa_pots.c
		cpg_elev_d						rwa_pots.c
		cpg_elev_c						rwa_pots.c
		cpg_elev_r						rwa_pots.c
		cpo_trav_l						rwa_pots.c
		cpo_trav_c						rwa_pots.c
		cpo_trav_r						rwa_pots.c
		cpo_elev_d						rwa_pots.c
		cpo_elev_c						rwa_pots.c
		cpo_elev_r						rwa_pots.c
		fclose						
	network_init							
	obj_create_objects							
	cig_prepare							
	buffer_setup							
	cig_synchronize							
	msg_startup							
	repair_uninit							
	hull_init							
	cig_stop							
	network_can_i_really_use_network							
	network_get_net_handle							
	filter_init							
	exit							

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
	get_priority_list_file							read_pars.c
		priority_list_file						read_pars.c
	rva_setup							
	sim_state_idle							main.c
		printf						
		sim_state						main.c
	veh_spec_startup							rwa_main.c
		rtc_init_clock						
		printf						
		network_set_simulator_type						
		use_cig_reconfig_startup						main.c
			cig_startup_func					main.c
			cig_reconfig_start					
		get_vconfig_file1						read_pars.c
			vconfig_file1					read_pars.c
		cig_set_view_config_file						
		get_ammo_map_file						read_pars.c
			ammo_map_file					read_pars.c
		map_file_read						
		get_veh_map_file						read_pars.c
			veh_map_file					read_pars.c
		map_vehicle_file_read						
		get_asid_map_file						read_pars.c
			asid_map_file					read_pars.c
		map_read_asid_file						
		init_activ						rwa_main.c
		rwa_config_init						rwa_config.c
			init_symbol_table					
			printf					
			reader_find_file					
			data_file					rwa_config.c
			exit					
			find_tag					
			vehicle_name					rwa_config.c
			munitions_table					rwa_config.c
			get_symbol					
			weapons_info_size					rwa_config.c
			weapons_info					rwa_config.c
			malloc					

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			bzero					
		keyboard_init						rwa_keybrd.c
			use_keybrd					rwa_keybrd.c
			input_char_count					rwa_keybrd.c
			entering_str					rwa_keybrd.c
			console_desc					rwa_keybrd.c
			keybrd_tty_init					sun_wayed.c
			printf					
			exit					
		failure_init						rwa_failure.c
			cfail_init					
			fail_table_init					
			get_sdamage_file					read_pars.c
				sdamage_file				read_pars.c
			MAINT_LEVEL_ARRAY					rwa_failure.c
			sfail_init					
		weapons_startup						rwa_weapons.c
			printf					
			air_veh_list_id					rwa_weapons.c
			stinger_is_air_veh					rwa_weapons.c
				is_air_vehicle				sun_stubs.c
			rva_create_output_list					sun_stubs.c
		ammo_resupply_init						rwa_ammo.c
			printf					
			rwa_ammo_resupply_list_id					rwa_ammo.c
			rwa_ammo_resupply_check					rwa_ammo.c
				is_friendly				sun_wayed.c
					vehicle_force			sun_wayed.c
				is_ammo_vehicle				
			rva_create_output_list					sun_stubs.c
			rwa_fuel_resupply_list_id					rwa_ammo.c
			rwa_fuel_resupply_check					rwa_ammo.c
				is_friendly				sun_wayed.c
					vehicle_force			sun_wayed.c
				is_fuel_vehicle				
			rwa_resupply_completed					rwa_ammo.c
				rwa_resupply_in progress				rwa_config.c
					resupply_in_progress			rwa_config.c
				rwa_config_determine_ammo_needed				rwa_config.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					resupply_in_progress			rwa_config.c
					hungry_for_ammo			rwa_config.c
					rwa_config_get_was_munition_index			rwa_config.c
						was		rwa_config.c
					resupply_get_ammo_offered			resupp.c
						ammo_offered		resupp.c
					leftwing_stores			rwa_config.c
					ammo_type_full			ammo.c
						ammo_struct		
					rightwing_stores			rwa_config.c
					turret_stores			rwa_config.c
				printf				
				rwa_config_get_was_munition_index				rwa_config.c
					was			rwa_config.c
				rwa_config_get_was_position_name				rwa_config.c
					was			rwa_config.c
				softp_label				
				leftwing_stores				rwa_config.c
				rightwing_stores				rwa_config.c
				turret_stores				rwa_config.c
			mun_set_veh_spec_resupply_completed					resupp.c
				veh_spec_resupply_completed				resupp.c
			rwa_resupply_started					rwa_ammo.c
				rwa_resupply_in progress				rwa_config.c
					resupply_in_progress			rwa_config.c
				printf				
				rwa_config_get_was_munition_index				rwa_config.c
					was			rwa_config.c
				rwa_config_get_was_position_name				rwa_config.c
					was			rwa_config.c
				softp_label				
				leftwing_stores				rwa_config.c
				rightwing_stores				rwa_config.c
				turret_stores				rwa_config.c
			mun_set_veh_spec_resupply_started					resupp.c
				veh_spec_resupply_started				resupp.c
		map_get_damage_files						
		use_intervisibility_server						rwa_main.c
		IV_CLIENT						

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
		Intervisibility Init						
		Intervisibility Synchronize						
	timers_simul							
	veh_spec_idle							rwa_main.c
		status_simul						rwa_status.c
			frame_counter					rwa_status.c
			monitor_status					rwa_status.c
				idc_values				
				hard_dead				
				fifo_hard				rwa_mem.c
				st_com				rwa_status.c
				fifo_enqueue				
				softi_dead				
				fifo_softi				
				softo_dead				
				ser_heratbeat				
				ser_dead				
				net_xmt_failed				
				net_dead				
				set_xmt_failed				
				dtad_failed				rwa_status.c
				dtad_dead				
				sound_dead				
				sounds				rwa_mem.c
				st_sound				rwa_status.c
				temperature				rwa_status.c
				current_temperature				
				voltage12P				rwa_status.c
				current_plus12				
				HILIMIT-12P				
				plus12_dead				
				LOLIMIT_12P				
				voltage12N				rwa_status.c
				current_minus12				
				HILIMIT_12N				
				minus12_dead				
				LOLIMIT_12N				
				voltage5				rwa_status.c
				current_plus5				

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
				HILIMIT_5				
				plus5_dead				
				LOLIMIT_5				
				need_to_set_host_red				rwa_status.c
				equipment_status				rwa_status.c
				need_to_set_cig_red				rwa_status.c
				need_to_set_hard_red				rwa_status.c
				need_to_set_softi_red				rwa_status.c
				need_to_set_softo_red				rwa_status.c
				need_to_set_sound_red				rwa_status.c
				need_to_set_voltage12P_red				rwa_status.c
				need_to_set_voltage12N_red				rwa_status.c
				need_to_set_voltage5_red				rwa_status.c
				need_to_set_net_red				rwa_status.c
				status_out				
		keyboard_simul						
		io_simul_idle						
		initial_activation						rwa_main.c
		need_to_fill_initial_munitions						rwa_main.c
		printf						
		rwa_config_initialize_munitions						rwa_config.c
			previous_vehicle_type					rwa_config.c
			data_file					rwa_config.c
			find_tag					
			printf					
			get_symbol					
		init_activ						rwa_main.c
		fill_vehicle_status						rwa_network.c
			fuel_get_current_level					fuelsys.c
				fuel_struct				fuelsys.c
			rwa_config_get_was_munition_type					rwa_config.c
				was				rwa_config.c
			rwa_config_get_was_munition_index					rwa_config.c
				was				rwa_config.c
			leftwing_stores					rwa_config.c
			ammo_check_availability					ammo.c
				ammo_index_ok				ammo.c
					printf			
			rightwing_stores					rwa_config.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			turret_stores					rwa_config.c
		network_get_exercise_id						
		process_activate_request						
	init_ballistics_buffer							
	idc_reset							
	veh_spec_init							rwa_main.c
		LRF Init						
		Lrf Err String						
		fprintf						
		fflush						
		softp_send_idc_reset						
		sound_reset						
		status_preset						
		firectl_init						
			firectl_was_init					
		controls_fsm_init						
		controls_sim_init						
		view_init						
		meter_init						
		resupply_init						
		rwa_init						rwa_simul.c
			view_point					rwa_simul.c
			kinematics_viewpoint_offset					
			engine_init					
				engine_power				rwa_engine.c
				engine_percent_torque				rwa_engine.c
				engine_speed				rwa_engine.c
				number_of_engines				rwa_engine.c
				engine_status				rwa_engine.c
				engine_is_damaged				rwa_engine.c
				transmission_is_damaged				rwa_engine.c
				starting_engine				rwa_engine.c
				integrator_gain				rwa_engine.c
				gov_p_gain				rwa_engine.c
				gov_i_gain				rwa_engine.c
				last_per_cent_shaft_torque				rwa_engine.c
				last_percent_torque				rwa_engine.c
				hours_of_flight				rwa_engine.c
				minutes_of_flight				rwa_engine.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
				old_minutes_of_flight				rwa_engine.c
				engine_damage_engine_oil				rwa_engine.c
					controls_start_failure_lamp_flashing			
					engine_is_damaged			rwa_engine.c
				engine_repair_engine_oil				rwa_engine.c
					controls_failure_lamp_off			
					engine_is_damaged			rwa_engine.c
				fail_init_failure				
				engine_break_engine				rwa_engine.c
					engine_status			rwa_engine.c
					engine_speed			rwa_engine.c
					number_of_engines			rwa_engine.c
				engine_repair_engine				rwa_engine.c
					engine_repair_engine_oil			rwa_engine.c
						controls_failure_lamp_off		
						engine_is_damaged		rwa_engine.c
					engine_status			rwa_engine.c
					number_of_engines			rwa_engine.c
				engine_damage_transmission_filter				
				engine_repair_transmission_filter				rwa_engine.c
					controls_failure_lamp_off			
					transmission_is_damaged			rwa_engine.c
				engine_break_transmission				rwa_engine.c
					engine_break_engine			rwa_engine.c
						engine_status		rwa_engine.c
						engine_speed		rwa_engine.c
						number_of_engines		rwa_engine.c
				engine_repair_transmission				rwa_engine.c
					engine_repair_transmission_filter			rwa_engine.c
						controls_failure_lamp_off		
						transmission_is_damaged		rwa_engine.c
					engine_repair_engine			rwa_engine.c
						engine_repair_engine_oil		rwa_engine.c
							controls_failure_lamp_off	
							engine_is_damaged	rwa_engine.c
						engine_status		rwa_engine.c
						number_of_engines		rwa_engine.c
			aerodyn_init					
				engine_init				

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					engine_power			rwa_engine.c
					engine_percent_torque			rwa_engine.c
					engine_speed			rwa_engine.c
					number_of_engines			rwa_engine.c
					engine_status			rwa_engine.c
					engine_is_damaged			rwa_engine.c
					transmission_is_damaged			rwa_engine.c
					starting_engine			rwa_engine.c
					integrator_gain			rwa_engine.c
					gov_p_gain			rwa_engine.c
					gov_i_gain			rwa_engine.c
					last_per_cent_shaft_torque			rwa_engine.c
					last_percent_torque			rwa_engine.c
					hours_of_flight			rwa_engine.c
					minutes_of_flight			rwa_engine.c
					old_minutes_of_flight			rwa_engine.c
					engine_damage_engine_oil			rwa_engine.c
						controls_start_failure_lamp_flashing		
						engine_is_damaged		rwa_
					engine_repair_engine_oil			rwa_
						controls_failure_lamp_off		
						engine_is_damaged		rwa_
					fail_init_failure			
					engine_break_engine			rwa_
						engine_status		rwa_
						engine_speed		rwa_
						number_of_engines		rwa_
					engine_repair_engine			rwa_
						engine_repair_engine_oil		rwa_
							controls_failure_lamp_off	
							engine_is_damaged	rwa_
						engine_status		rwa_
						number_of_engines		rwa_
					engine_damage_transmission_filter			
					engine_repair_transmission_filter			rwa_
						controls_failure_lamp_off		
						transmission_is_damaged		rwa_
					engine_break_transmission			rwa_
						engine_break_engine		rwa_

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th			
							engine_status			rwa_
							engine_speed			rwa_
							number_of_engines			rwa_
					engine_repair_transmission					rwa_
						engine_repair_transmission_filter			rwa_engine.c	
							controls_failure_lamp_off			
							transmission_is_damaged		rwa_engine.c	
						engine_repair_engine				rwa_
							engine_repair_engine_oil		rwa_engine.c	
								controls_failure_lamp_off		
								engine_is_damaged	rwa_engine.c	
							engine_status			rwa_
							number_of_engines			rwa_
				sin						
				cos						
				vec_init						
				ground_force						
				vehicle_mass_init						
				ground_init						
				find_cubic_func						
				fprintf						
				get_constants_file						
				aerodyn_read_simple_constants						
					fopen					
					printf					
					fgets					
					strtok					
					strcmp					
					sscanf					
					fclose					
			rwa_config_get_front_support						rwa_config.c	
				front_support					rwa_config.c	
			aero_body_point_set_front_wheels						rwa_aerodyn.c	
				body_point					rwa_ground.h	
				ground_height					rwa_aerodyn.c	
				printf						
			rwa_config_get_rear_support						rwa_config.c	
				rear_support					rwa_config.c	
			aero_body_point_set_rear_wheel						rwa_aerodyn.c	

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
				body_point				rwa_ground.h
				printf				
		alt_init						
		gunmnt_init						
		tads_init						
		sad_uninit						
		SAD_TTY_PORT						
		sad_init						
		ammo_configuration_menu_init						
		weapons_init						rwa_weapons.c
			gunmnt_element					rwa_weapons.c
			hull					
			rotate_init_element					
			cig_set_proc_hit_msg					rwa_cig.c
				rwa_process_msg_hit_return				rwa_cig.c
					f2d_vec_copy			
					map_is_missile			
					missile_util_comm_intersected_poly			util_comm.c
						missile_comm		util_comm.c
					msg_get_veh_id_from_cig_id			
					missile_util_comm_intersected_model			util_comm.c
						missile_comm		util_comm.c
					proc_hit_debug			rwa_cig.c
					printf			
					map_is_bomb			
					veh_kinematics			sun_stubs.c
					kinematics_range_squared			
					map_get_network_type_from_ammo_entry			sun_stubs.c
					network_ifire_init_burst			
					network_ifire_send_detonation			
					network_ifire_send_indirect_fire			
					impacts_queue_effect			
					rwa_config_get_was_munition_index			
					rounds_interp_rounds			
					cig_impact_from_round_fired			
				change_process_msg_hit_function				
			rounds_init_volley					rwa_rounds.c
				volley_list				rwa_rounds.c
				volley_free				rwa_rounds.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
				free_ptr				fuze_prox.c
				first_volley				rwa_rounds.c
				last_volley				rwa_rounds.c
			rocket_laser_range					rwa_weapons.c
			gun_firing_state					rwa_weapons.c
			new_gun_firing_state					rwa_weapons.c
			gun_switch					rwa_weapons.c
			gun_failure					rwa_weapons.c
			ATGM_launch_from_left					rwa_weapons.c
			stinger_launch_from_left					rwa_weapons.c
			left_stores_position					rwa_weapons.c
			right_stores_position					rwa_weapons.c
			hydra_init.c					rwa_hydra.c
				articulation_element				rwa_hydra.c
				hull				
				rotate_init_element				
				printf				
				pylon_L_element				rwa_hydra.c
				articulation				rwa_hydra.c
					articulation_element			rwa_hydra.c
				pylon_R_element				rwa_hydra.c
				hydras				rwa_hydra.c
				missile_hydra_init				rkt_hydra.c
					hydra_array			rkt_hydra.c
					num_hydra			rkt_hydra.c
					rkts_in_flight			rkt_hydra.c
					hydra_fly			rkt_hydra.c
					pylon_x			rkt_hydra.c
					pylon_y			rkt_hydra.c
					pylon_z			rkt_hydra.c
					flight_time			rkt_hydra.c
					speed_factor			miss_hellfr.c
					max_range_limit			miss_hellfr.c
					ball_table_loaded			rkt_hydra.c
					printf			
					table_size			rkt_hydra.c
					ball_table			rkt_hydra.c
					missile_util_load_ball_traj_file			util_ball.c
						fopen		

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
						fprintf		
						exit		
						eof		ball_load.c
							getc	
							ungetc	
						fscanf		
						fclose		
						printf		
					flechette_veh_list			rkt_hydra.c
					flechette_is_valid_veh			sub_flech.c
					rva_create_output_list			sun_stubs.c
					missile_fuze_prox_init			fuze_prox.c
						prox_list		fuze_prox.c
						prox_free		fuze_prox.c
						free_ptr		fuze_prox.c
			tows					rwa_weapons.c
			missile_tow_init					miss_tow.c
				speed_factor				miss_hellfr.c
				max_range_limit				miss_hellfr.c
				max_range_squared				miss_hellfr.c
				tow_ammo_type				miss_tow.c
			hellfires					rwa_weapons.c
			missile_hellfire_init					miss_hellfr.c
				speed_factor				miss_hellfr.c
				max_range_limit				miss_hellfr.c
				max_range_squared				miss_hellfr.c
				hellfire_ammo_type				miss_hellfr.c
			stingers					rwa_weapons.c
			missile_stinger_init					miss_stinger.c
				num_stingers				miss_stinger.c
				stinger_array				miss_stinger.c
				speed_factor				miss_hellfr.c
				max_range_limit				miss_hellfr.c
				max_range_squared				miss_hellfr.c
				stinger_ammo_type				miss_stinger.c
				missile_fuze_prox_init				fuze_prox.c
					prox_list			fuze_prox.c
					prox_free			fuze_prox.c
					free_ptr			fuze_prox.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			weapons_config_missile					rwa_weapons.c
				rwa_config_get_was_munition_info				rwa_config.c
					was			rwa_config.c
				missile_hellfire_set_ammo_type				miss_hellfr.c
					hellfire_ammo_type			miss_hellfr.c
				missile_hellfire_set_max_range_limit				miss_hellfr.c
					max_range_limit			miss_hellfr.c
					max_range_squared			miss_hellfr.c
				missile_hellfire_set_speed_factor				miss_hellfr.c
					speed_factor			miss_hellfr.c
				missile_stinger_set_ammo_type				miss_stinger.c
					stinger_ammo_type			miss_stinger.c
				missile_stinger_set_max_range_limit				miss_stinger.c
					max_range_limit			miss_hellfr.c
					max_range_squared			miss_hellfr.c
				missile_stinger_set_speed_factor				miss_stinger.c
					speed_factor			miss_hellfr.c
				missile_tow_set_ammo_type				miss_tow.c
					tow_ammo_type			miss_tow.c
				missile_tow_set_max_range_limit				miss_tow.c
					max_range_limit			miss_hellfr.c
					max_range_squared			miss_hellfr.c
				missile_tow_set_speed_factor				miss_tow.c
					speed_factor			miss_hellfr.c
				printf				
				missile_util_init				util_init.c
					missile_util_comm_init			util_comm.c
						missile_comm		util_comm.c
						network_missiles_init		
			weapons_break_gun_major					
			weapons_repair_gun_major					
			fail_init_failure					
			weapons_break_gun					
			weapons_repair_gun					
			weapons_break_hellfire					rwa_weapons.c
				controls_start_failure_lamp_flashing				
			weapons_repair_hellfire					rwa_weapons.c
				controls_failure_lamp_off				
			weapons_break_stinger					rwa_weapons.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
				controls_start_failure_lamp_flashing				
			weapons_repair_stinger					rwa_weapons.c
				controls_failure_lamp_off				
		bcs_init						
		vision_restore_all_blocks						
		controls_edge_unit						
		app_init						
		cig_2d_do_init						
		controls_copil_recalib						
		sad_show_aircraft						
			rad_meter_preset					
		veh_kinematics						sun_stubs.c
		kinematics_get_o_to_h						sun_stubs.c
		kinematics_get_w_to_h						
		config_pos_init2						
		cig_init_ctr						rwa_cig.c
			init_cig_ticks					rwa_cig.c
		HET_TTY_PORT						
		get_het_calib_file						read_pars.c
			het_calib_file					read_pars.c
		head_eye_tracker_init						
		head_eye_tracker_send_request						
		sight						
		view						rwa_view.c
			view_element					rwa_view.c
		view_attacker_in_fov						
			view					
		tads_vehicle_in_fov						
			sight					
		network_get_net_handle						
		network_current_time_in_ms						
		het_init						
		laserdam_init						
	impacts_init							
	turret_init							rwa_rotate.c
		veh_spec_kinematics_init						rwa_kinemat.c
	repair_init							rwa_stubs.c
	imers_init_starttime							
	rva_init							

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
	msg_init							
	obj_init_objects							
	cig_startup_func							main.c
	cig_startup_func_FPTR							main.c
	buffer_reset							
	cig_spec_init							rwa_cig.c
		cig_msg_prepend_request_laser_range						
	fail_init							
	sim_state_simulate							main.c
		printf						
		sim_state						main.c
	RTC_FRAME							
	rtc_start_time							
	RTC_FRAME_GAP							
	bbd_bit_out							
	RTC_TIMERS_SIMUL							
	rtc_stop_time							
	RTC_FAIL_SIMUL							
	fail_simul							
	RTC_VEH_SPEC_SIMUL							
	veh_spec_simulate							rwa_main.c
		status_simul						rwa_status.c
			frame_counter					rwa_status.c
			monitor					
		keyboard_simul						
		waypoint_editor						
		sad_simul						
		sound_simul						rwa_sound.c
			sound_error					rwa_sound.c
		controls_simul						rwa_ctl_fsm.c
			controls_status					rwa_ctl_fsm.c
			controls_sim_next_state					rwa_ctl_fsm.c
				controls_failure_val				rwa_ctl_fsm.c
				controls_sim_routines				rwa_ctl_sim.c
					controls_pil_cyc_roll_check			
					controls_pil_cyc_pitch_check			
					controls_pil_pedal_check			
					controls_pil_coll_check			
					controls_copil_trav_check			

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					controls_copil_elev_check			
					controls_pil_trigger_1_check			
					controls_pil_trigger_2_check			
					controls_cpg_trigger_1_check			
					controls_cpg_trigger_2_check			
					controls_cpg_sensor_select_check			
					controls_copil_laser_burst_check			
					controls_cpg_cont_laser_check			
					controls_laser_master_check			
					controls_weapons_master_check			
					controls_weapons_cpg_check			
					controls_view_slew_check			
					controls_pil_was_check			
					controls_cpg_was_check			
					controls_target_store_check			
					controls_cpo_auto_track_check			
					controls_slave_check			
					controls_cpg_auto_track_toggle_check			
					controls_hover_hold_check			
					controls_wide_fov_check			
					controls_narrow_fov_check			
					controls_zoom_fov_check			
					controls_medium_fov_check			
					controls_cpo_sensor_check			
					controls_polarity_check			
					controls_radar_warning_flash_check			
					controls_failure_lamp_flash_check			
					controls_master_caution_check			
					controls_manual_range_check			
				controls_failure_edge				rwa_ctl_fsm.c
				controls_sim_off				
			fprintf					
			nprintf					
		view_simul						
		ammo_simul						rwa_ammo.c
			ammo_quantity_has_changed					ammo.c
				ammo_indicators_require_updating				ammo.c
			rwa_config_get_was_munition_index					rwa_config.c
			leftwing_stores					

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			ammo_check_availability					
			rightwing_stores					rwa_config.c
			turret_stores					rwa_config.c
			meter_missile1_set					
			meter_missile2_set					
			meter_rocket_set					
			meter_ammo_set					
			resupply_receive_gating_conditions_ok					
			rva_lists_off					rwa_ammo.c
			rwa_ammo_resupply_list_id					rwa_ammo.c
			rva_build_list					
			rwa_fuel_resupply_list_id					rwa_ammo.c
			rva_get_output_list					sun_stubs.c
			rwa_config_determine_ammo_needed					
			mun_set_ammo_resupply_list					
			mun_set_fuel_resupply_list					
			rva_dont_build_list					
			resupply_simul					
		fuel_simul						
		meter_simul						rwa_meter.c
		resupply_simul						
		bbd_bit_out						
		RTC_RWA_SIMUL						
		rtc_start_time						
		rwa_simul						rwa_simul.c
			get_selected_model					
			aerodyn_simul					rwa_aerodyn.c
				get_aircraft_kinematic_state				
					orientation_calc			
					parameters_calc			
					true_airspeed			rwa_aerodyn.c
					kinematics_get_true_airspeed			rwa_kinemat.c
						true_airspeed		rwa_aerodyn.c
					altitude			rwa_aerodyn.c
					kinematics_get_altitude			rwa_kinemat.c
						altitude		rwa_aerodyn.c
					angular_velocity_vector			rwa_aerodyn.c
					kinmatics_get_angular_velocity_vector			rwa_kinemat.c
						ang_vel		rwa_kinemat.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					normalized_velocity_vector			
					kinematics_get_normalized_velocity_vector			rwa_kinemat.c
						true_airspeed		rwa_kinemat.c
						norm_vel		rwa_kinemat.c
						pos_unit_vel		rwa_kinemat.c
						neg_unit_vel		rwa_kinemat.c
					velocity_vector			rwa_aerodyn.c
					kinematics_get_linear_velocity_vector			rwa_kinemat.c
						velocity_vector		rwa_aerodyn.c
					gravity_dir_vector			rwa_aerodyn.c
					kinematics_get_gravity_vector			rwa_kinemat.c
						gravity		rwa_kinemat.c
					angle_of_attack			rwa_aerodyn.c
					kinematics_get_aoa			
					side_slip_angle			rwa_aerodyn.c
					kinematics_get_yaw			
					velocity_to_body			rwa_aerodyn.c
					kinematics_get_velocity_to_body			rwa_kinemat.c
						velocity_to_body		rwa_aerodyn.c
					g_force			rwa_aerodyn.c
					kinematics_get_g_force			rwa_kinemat.c
						g_force		rwa_aerodyn.c
					vertical_speed			rwa_aerodyn.c
					kinematics_get_vertical_speed			rwa_kinemat.c
						vertical_speed		rwa_aerodyn.c
				compute_flight_parameters				rwa_aerodyn.c
					ambient_density			rwa_aerodyn.c
					altitude			rwa_aerodyn.c
					air_density			
					ambient_temperature			rwa_aerodyn.c
					air_temperature			
					ambient_pressure			rwa_aerodyn.c
					air_pressure			
					dynamic_pressure			rwa_aerodyn.c
					true_airspeed			rwa_aerodyn.c
					pitch_rate			rwa_aerodyn.c
					angular_velocity_vector			rwa_aerodyn.c
					roll_rate			rwa_aerodyn.c
					yaw_rate			rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					roll			rwa_aerodyn.c
					gravity_dir_vector			rwa_aerodyn.c
					atan2			
					pitch			rwa_aerodyn.c
				compute_stab_augmentation_gains				rwa_aerodyn.c
					hover_hold_state			rwa_aerodyn.c
					hover_hold_turned_on			rwa_aerodyn.c
					pitch_damping			rwa_aerodyn.c
					roll_damping			rwa_aerodyn.c
					hover_aug_roll_integrator			rwa_aerodyn.c
					hover_aug_pitch_integrator			rwa_aerodyn.c
					stab_aug_yaw_integrator			rwa_aerodyn.c
					stab_aug_climb_integrator			rwa_aerodyn.c
					true_airspeed			rwa_aerodyn.c
					MAX_ATT_CTL_ANGLE			rwa_aerodyn.c
					log			
					velocity_vector			rwa_aerodyn.c
					limiter			rwa_aerodyn.c
					hover_aug_roll_angle			rwa_aerodyn.c
					stab_aug_roll			rwa_aerodyn.c
					set_roll_attitude			rwa_aerodyn.c
						attitude_control_roll_integrator		rwa_aerodyn.c
						roll		rwa_aerodyn.c
						limiter		rwa_aerodyn.c
						attitude_control_roll_command		rwa_aerodyn.c
					hover_aug_pitch_angle			rwa_aerodyn.c
					stab_aug_pitch			rwa_aerodyn.c
					set_pitch_attitude			rwa_aerodyn.c
						attitude_control_pitch_integrator		rwa_aerodyn.c
						pitch		rwa_aerodyn.c
						limiter		rwa_aerodyn.c
						attitude_control_pitch_command		rwa_aerodyn.c
					angular_velocity_vector			rwa_aerodyn.c
					stab_aug_yaw			rwa_aerodyn.c
					stab_aug_climb			rwa_aerodyn.c
					controller_cyclic_roll			rwa_aerodyn.c
					cyclic_roll			rwa_aerodyn.c
					controller_cyclic_pitch			rwa_aerodyn.c
					cyclic_pitch			rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					controller_tail_rotor			rwa_aerodyn.c
					pedal			rwa_aerodyn.c
					controller_collective			rwa_aerodyn.c
					collective			rwa_aerodyn.c
				compute_rotor_loads				rwa_aerodyn.c
					main_rotor_load_torque			rwa_aerodyn.c
					controller_collective			rwa_aerodyn.c
					tail_rotor_load_torque			rwa_aerodyn.c
					controller_tail_rotor			rwa_aerodyn.c
				compute_engine_torque				rwa_aerodyn.c
					main_rotor_load_torque			rwa_aerodyn.c
					tail_rotor_load_torque			rwa_aerodyn.c
					altitude			rwa_aerodyn.c
					engine_simul			rwa_engine.c
						engine_load_torque		rwa_engine.c
						engine_power		rwa_engine.c
						gov_p_gain		rwa_engine.c
						engine_speed		rwa_engine.c
						engine_status		rwa_engine.c
						integrator_gain		rwa_engine.c
						gov_i_gain		rwa_engine.c
						fuel_level_empty		fuelsys.c
							fuel_struct	fuelsys.c
						engine_drive_torque		rwa_engine.c
						number_of_engines		rwa_engine.c
						engine_percent_torque		rwa_engine.c
						turbine_speed		rwa_engine.c
						main_rotor_shaft_speed		rwa_engine.c
						tail_rotor_shaft_speed		rwa_engine.c
						powertrain_percent_shaft_speed		rwa_engine.c
						tail_rotor_drive_torque		rwa_engine.c
						main_rotor_drive_torque		rwa_engine.c
						fuel_flow		rwa_engine.c
						sound_stop_cont_sound		
						starting_engine		rwa_engine.c
						fuel_used_by_engine		fuelsys.c
							fuel_struct	fuelsys.c
							fuel_indicators_require_updating	fuelsys.c
						meter_torque_set		rwa_meter.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th		
							controls_power_status		rwa_ctl_fsm.c
								controls_status	rwa_ctl_fsm.c
							controls_failure_status		rwa_ctl_fsm.c
								controls_failure_val	rwa_ctl_fsm.c
							conv_frac_to_percent		rwa_meter.c
							torque_set_val		rwa_meter.c
							torque_oscillation		rwa_meter.c
							softp_ins_panel_set		
						meter_rpm_set			rwa_meter.c
							controls_power_status		rwa_ctl_fsm.c
								controls_status	rwa_ctl_fsm.c
							controls_failure_status		rwa_ctl_fsm.c
								controls_failure_val	rwa_ctl_fsm.c
							conv_frac_to_percent		rwa_meter.c
							rpm_set_val		rwa_meter.c
							softp_ins_panel_set		
						hours_of_flight			rwa_engine.c
						minutes_of_flight			rwa_engine.c
						old_minutes_of_flight			rwa_engine.c
						sfail_event_occurred			
						engine_is_damaged			rwa_engine.c
						transmission_is_damaged			rwa_engine.c
						engine_sound_type			rwa_engine.c
						last_percent_shaft_speed			rwa_engine.c
						sound_make_cont_sound			
						last_percent_torque			rwa_engine.c
						rotor_oscillation			rwa_engine.c
						sound_make_arg_sound			
						engine_oscillation			rwa_engine.c
					powertrain_percent_shaft_speed				rwa_aerodyn.c
					engine_get_rotor_percent_shaft_speed				rwa_engine.c
						[pwertrain_percent_shaft_speed			rwa_aerodyn.c
				compute_rotor_forces_and_moments					rwa_aerodyn.c
					main_rotor_thrust				rwa_aerodyn.c
					powertrain_percent_shaft_speed				rwa_aerodyn.c
					controller_collective				rwa_aerodyn.c
					tail_rotor_thrust				rwa_aerodyn.c
					controller_tail_rotor				rwa_aerodyn.c
					force_body_main_rotor				rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					MAIN_ROTOR_MAST_TILT_SIN			rwa_aerodyn.c
					MAIN_ROTOR_MAST_TILT_COS			rwa_aerodyn.c
					force_body_tail_rotor			rwa_aerodyn.c
					moment_body_main_rotor			rwa_aerodyn.c
					controller_cyclic_pitch			rwa_aerodyn.c
					controller_cyclic_roll			rwa_aerodyn.c
					main_rotor_torque_load			rwa_aerodyn.c
				compute_lift_drag_coefficients				rwa_aerodyn.c
					lift_coefficient_vstab			rwa_aerodyn.c
					side_slip_angle			rwa_aerodyn.c
					vstab_lift_coefficient			rwa_aerodyn.c
					lift_coefficient_virtual_wing			rwa_aerodyn.c
					true_airspeed			rwa_aerodyn.c
					p_drag_fit_coeff			rwa_aerodyn.c
					cubic_func			
					angle_of_attack			rwa_aerodyn.c
					sin			
					total_incompressible_drag_coefficient			rwa_aerodyn.c
				compute_lift_drag_forces				rwa_aerodyn.c
					lift_virtual_wing			rwa_aerodyn.c
					dynamic_pressure			rwa_aerodyn.c
					lift_coefficient_virtual_wing			rwa_aerodyn.c
					lift_vstab			rwa_aerodyn.c
					lift_coefficient_vstab			rwa_aerodyn.c
					total_drag			rwa_aerodyn.c
					total_incompressible_drag_coefficient			rwa_aerodyn.c
				compute_body_damping_forces_and_moments				rwa_aerodyn.c
					moment_body_damping			rwa_aerodyn.c
					pitch_damping			rwa_aerodyn.c
					ptich_rate			rwa_aerodyn.c
					roll_damping			rwa_aerodyn.c
					roll_rate			rwa_aerodyn.c
					yaw_damping			rwa_aerodyn.c
					yaw_rate			rwa_aerodyn.c
					force_body_damping			rwa_aerodyn.c
					velocity_vector			rwa_aerodyn.c
				transform_lift_drag_forces_to_body_coordinates				rwa_aerodyn.c
					virtual_wing_force			rwa_aerodyn.c
					lift_virtual_wing			rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					vstab_force			rwa_aerodyn.c
					lift_vstab			rwa_aerodyn.c
					drag_force			rwa_aerodyn.c
					total_drag			rwa_aerodyn.c
					true_airspeed			rwa_aerodyn.c
					pitch			rwa_aerodyn.c
					sin			
					velocity_to_body			rwa_aerodyn.c
					lift_body_virtual_wing			rwa_aerodyn.c
					vec_mat_mul			
					lift_body_vstab			rwa_aerodyn.c
					drag_body			rwa_aerodyn.c
				generate_gravity_body_force				rwa_aerodyn.c
					compute_gross_weight			rwa_aerodyn.c
						vehicle_mass		rwa_aerodyn.c
						fuel_get_current_level		fuelsys.c
							fuel_struct	fuelsys.c
						gross_weight		rwa_aerodyn.c
					gravity_force_body			rwa_aerodyn.c
					gravity_dir_vector			rwa_aerodyn.c
					gross_weight			rwa_aerodyn.c
				interact_with_ground				rwa_aerodyn.c
					normalized_velocity_vector			rwa_aerodyn.c
					true_airspeed			rwa_aerodyn.c
					body_point			rwa_ground.h
					ground_force			
					ground_torque			
					grnd			rwa_ground.h
					ground_interaction			
					force_ground_effect			rwa_aerodyn.c
					main_rotor_thrust			rwa_aerodyn.c
					cig_altitude_above_gnd			sun_wayed.c
				sum_body_forces_and_moments_about_ac				rwa_aerodyn.c
					force_body			rwa_aerodyn.c
					vec_init			
					force_body_main_rotor			rwa_aerodyn.c
					vec_add			
					lift_body_virtual_wing			rwa_aerodyn.c
					lift_body_vstab			rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					drag_body			rwa_aerodyn.c
					force_body_damping			rwa_aerodyn.c
					gravity_force_body			rwa_aerodyn.c
					ground_force			
					force_ground_effect			rwa_aerodyn.c
					loc_ac_tail_rotor_cop			rwa_aerodyn.c
					force_body_tail_rotor			rwa_aerodyn.c
					moment_body_tail_rotor			rwa_aerodyn.c
					vec_cross_prod			
					loc_ac_virtual_wing_cop			rwa_aerodyn.c
					moment_body_virtual_wing			rwa_aerodyn.c
					loc_ac_vstab_cop			rwa_aerodyn.c
					moment_body_vstab			rwa_aerodyn.c
					loc_ac_cg			rwa_aerodyn.c
					moment_body_cg			rwa_aerodyn.c
					moment_body			rwa_aerodyn.c
					moment_body_main_rotor			rwa_aerodyn.c
					ground_torque			
					moment_body_damping			rwa_aerodyn.c
				send_to_dynamics_kinematics				rwa_aerodyn.c
					vehicle_mass			rwa_aerodyn.c
					inertia_matrix			rwa_aerodyn.c
					vehicle_mass_init			
					force_body			rwa_aerodyn.c
					vehicle_forces			
					moment_body			rwa_aerodyn.c
					vehicle_torques			
				vehicle_update				
			aerodyn_simple_simul					
			aerodyn_stealth_simul					
			printf					
			bbd_bit_out					
			kinematics_get_roll					rwa_kinemat.c
				roll				rwa_aerodyn.c
			fabs					
			current_bank					rwa_simul.c
			sound_make_const_sound					
			kinematics_get_body_pitch					rwa_kinemat.c
				body_pitch				rwa_kinemat.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			meter_adi_set					
			kinematics_get_heading					
			tads					rwa_tads.c
				tads_element				rwa_tads.c
			rotate_sight_angle					
			sight					
			laser_range					
			get_cmd_heading_state					
			meter_dg_set					
			get_cmd_heading					rwa_meter.c
				cmd_heading_val				rwa_cig_2d.c
			cig_altitude_above_gnd					sun_wayed.c
			meter_radar_alt_set					
			veh_kinematics					sun_stubs.c
			kinematics_get_d_pos					
			world					
			hull					
			rotate_get_mat					
			vec_mat_mul					
			meter_ball_set					
			softp_send_end_of_tick					
			cig_2d_set_dg_heading					rwa_cig_2d.c
				dg_heading_val				rwa_cig_2d.c
			cig_2d_set_alt_sen_bearing					rwa_cig_2d.c
				alt_sen_bearing_val				rwa_cig_2d.c
			cig_2d_set_laser_range					rwa_cig_2d.c
				laser_range_val				rwa_cig_2d.c
			cig_2d_set_azimuth					rwa_cig_2d.c
				azimuth_val				rwa_cig_2d.c
			cig_2d_set_elevation					rwa_cig_2d.c
				elevation_val				rwa_cig_2d.c
		rtc_stop_time						
		tads_simul						
		firectl_simul						
		weapons_simul						rwa_weapons.c
			automatic_gun_simul					rwa_weapons.c
				firectl_gun_selected				rwa_firectl.c
				tads_currently_fixed_forward				rwa_tads.c
				firectl_gun_selected_by_pilot				rwa_firectl.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					pil_weapon_select_state			rwa_firectl.c
				bcs_turn_computer_off				rwa_
				bcs_set_ballistics_computer				rwa_weapons.c
					super_elevation			rwa_hydra.c
					yb			rwa_weapons.c
					zb			rwa_weapons.c
					bcs_range			rwa_weapons.c
					ballistics_calc_se			ball_calc.c
						ballistics_calc_time		bal_calc.c
							sqrt	
							fprintf	
						atan		
				gun_out_of_contstraints				rwa_wapons.c
				cig_2d_set_status_message				rwa_cig_2d.c
				new_gun_firing_state				rwa_weapons.c
				tads				rwa_tads.c
				rotate_get_angle				
				bias_vector				rwa_weapons.c
				sight				rwa_tads.c
				bcs_get_super_elevation				rwa_weapons.c
					super_elevation			rwa_hydra.c
				gun_limits				rwa_weapons.c
					firectl_rocket_selected			rwa_firectl.c
						cpg_weapon_select_state		rwa_firectl.c
						pil_weapon_select_state		rwa_firectl.c
					fabs			
				gun_switch				rwa_weapons.c
				new_shot				rwa_weapons.c
				shot_counter				rwa_weapons.c
				shot_interval				rwa_weapons.c
				last_shot				rwa_weapons.c
				ammo_type				rwa_hydra.c
				turret_stores				rwa_config.c
				ammo_check_availability				
				leftwing_stores				rwa_config.c
				rightwing_stores				rwa_config.c
				tracer_round_interval				rwa_weapons.c
				gun_impacts_per_round				rwa_weapons.c
				weapons_fire_round				rwa_weapons.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					fire_g_to_w_mat			rwa_weapons.c
					continuous_gun_sounds			rwa_weapons.c
					gun_impacts_per_round			rwa_weapons.c
					sound_make_const_sound			
					veh_kinematics			sun_stubs.c
					kinematics_get_d_pos			
					vec_scale			
					d2f_vec_copy			
					event_get_event			
					bias_vector			rwa_weapons.c
					projectile_drift			rwa_weapons.c
					scaled_rand			
					tads_currently_fixed_forward			rwa_tads.c
					firectl_gun_selected_by_pilot			rwa_firectl.c
					gun_munition_data			rwa_weapons.c
					world			
					gunmnt			rwa_weapons.c
						gunmnt_element		rwa_weapons.c
					rotate_get_loc			
					fixed_gun			rwa_tads.c
						fixed_gun_element		rwa_tads.c
					rotate_get_mat			
					sight			rwa_tads.c
					gunmnt_get_sight_to_world			rwa_gunmnt.c
						bcs_computer_status		rwa_weapons.c
							bcs_booted_up	rwa_weapons.c
						bcs_get_super_elevation		rwa_weapons.c
						mat_rot)init2		
						mat_mat_mul		
					map_get_tracer_from_ammo_entry			
					ballistics_fire_a_round			ball_fire.c
					rounds_update_last_volley			rwa_rounds.c
						last_volley		rwa_rounds.c
					network_can_i_really_use_network			
					null_vehicleID			rkt_hydra.c
					network_send_shell_fire_pkt			
				ammo_fired				
				rounds_get_volley				rwa_rounds.c
				continuous_gun_sounds				rwa_weapons.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
				gun_firing_state				rwa_weapons.c
				make_sound_const_sound				rwa_weapons.c
			missile_simul					rwa_weapons.c
				new_reticle_state				rwa_weapons.c
				current_reticle_state				rwa_weapons.c
				reticle_on				
				sound_make_const_sound				rwa_cig_2d.c
				cig_2d_set_inner_box				rwa_tads.c
				sight				
				hull				
				rotate_get_mat				
				vec_mat_mul				rwa_weapons.c
				stinger_searching				rwa_weapons.c
				current_search_state				rwa_cig_2d.c
				cig_2d_set_stinger_location				rwa_weapons.c
				air_veh_list_id				
				rva_dont_build_list				rwa_weapons.c
				stinger_ready				
				rva_build_list				
				world				
				rotate_get_loc				miss_stinger.c
				missile_stinger_pre_launch				
					near_get_preferred_veh_near_vector			targ_pursuit.c
					missile_target_pursuit			
						vec_copy		
				printf				rwa_weapons.c
				tows				miss_tow.c
				missile_tow_fly				miss_hellfr.c
					max_range_limit			sun_stubs.c
					veh_kinematics			
					kinematics_range_squared			miss_hellfr.c
					max_range_squared			miss_hellfr.c
					speed_factor			miss_atgm.c
					tow_burn_speed_coeff			util_eval.c
					missile_util_eval_poly			miss_atgm.c
					toe_burn_turn_coeff			util_eval.c
					missile_util_eval_cos_coeff			util_eval.c
						missile_util_eval_poly		miss_atgm.c
					tow_coast_speed_coeff			

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					tow_coast_turn_coeff			miss_atgm.c
					missile_target_ground			targ_ground.c
					missile_target_level_los			targ_lev_los.c
					missile_util_flyout			
					missile_tow_stop			miss_tow.c
						missile_util_comm_stop_missile		
					missile_util_comm_check_intersection			
					missile_util_comm_check_detonate			
				hellfires				rwa_weapons.c
				laser_point				miss_hellfr.c
				missile_hellfire_fly				miss_hellfr.c
					speed_factor			miss_hellfr.c
					hellfire_burn_speed_coeff			util_eval.c
					missile_util_eval_poly			miss_hellfr.c
					hellfire_coast_speed_coeff			
					sqrt			
					cos			
					max_range_limit			miss_hellfr.c
					veh_kinematics			sun_stubs.c
					kinematics_range_squared			
					max_range_squared			miss_hellfr.c
					missile_target_ground			targ_ground.c
					missile_target_agm			targ_agm.c
						agm_seek		targ_agm.c
							sqrt	
							vec_scale	
							vec_add	
						vec_sub		
						vec_copy		
						sqrt		
						vec_dot_prod		
						vec_scale		
						vec_add		
					missile_util_flyout			util_flyout.c
						vec_sub		
						vec_dot_prod		
						vec_copy		
						sqrt		
						vec_scale		

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th		
						vec_add			
						missile_util_comm_fly_missile			util_comm.c
							printf		
							veh_kinematics		sun_stubs.c
							kinematics_range_squared		
							missile_comm		util_comm.c
							map_get_get_tracer_from_ammo_entry		
							store_traj_chord		
							network_send_missile_appearance		
							missile_util_comm_stop_missile		util_comm.c
								printf	
								missile_comm	util_comm.c
								network_send_non_impact	
								network_stop_missile_flyout	
					missile-hellfire_stop				miss_hellfr.c
						missile_util_comm_stop_missile			util_comm.c
					missile_util_comm_check_intersection				util_comm.c
					missile_util_comm_check_detonate				
				cig_2d_check_tof_countdown					rwa_cig_2d.c
				firectl_hellfire_selected					rwa_firectl.c
					pil_weapon_select_state				rwa_firectl.c
					pil_was_position				rwa_firectl.c
					rwa_config_get_was_munition_info				rwa_config.c
					cpg_weapon_select_state				rwa_firectl.c
					cpg_was_position				rwa_firectl.c
				cig_2d_check_tof_countdown_msg					rwa_cig_2d.c
				is_printing_tof					rwa_cig_2d.c
					print_tof				
				laser_range					
				cig_2d_set_tof					
				missile_hellfire_calc_tof					
				missile_stinger_fly_missiles					miss_stinger.c
					num_stingers				miss_stinger.c
					stinger_array				miss_stinger.c
					missile_stinger_fly				miss_stinger.c
						stinger_burn_speed_coeff			miss_stinger.c
						missile_util_eval_poly			util_eval.c
						stinger_coast_speed_coeff			miss_stinger.c
						sqrt			

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
						cos		
						near_get_preferred_veh_near_vector		
						max_range_limit		miss_hellfr.c
						veh_kinematics		sun_stubs.c
						kinematics_range_squared		
						max_range_squared		miss_hellfr.c
						missile_target_ground		targ_ground.c
						missile_target_intercept_pre_burnout		
						missile_target_intercept		
						missile_target_unguided		targ_unguide.c
						missile_util_flyout		
						missile_stinger_stop		
						missile_fuze_prox		
						missile_util_comm_check_detonate		
			rounds_check_volleys					rwa_rounds.c
				first_volley				rwa_rounds.c
				last_volley				rwa_rounds.c
				rounds_free_volley				rwa_rounds.c
					free_ptr			fuze_prox.c
					printf			
					volley_free			rwa_rounds.c
					free			
		head_eye_tracker_receive_data						
		head_eye_tracker_send_request						
		Lrf Tick						
		Lrf Err String						
		fprintf						
	RTC_KINEMATICS_SIMUL							
	veh_kinematics							sun_stubs.c
	kinematics_simul							
	RTC_TURRET_SIMUL							
	turret_simul							rwa_rotate.c
		rotate_hull_simul						
		rotate_simul						
		veh_spec_kinematics_simul						rwa_kinemat.c
			world					
			hull					
			rotate_get_loc					
			altitude					rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			velocity_vector					rwa_aerodyn.c
			true_airspeed					rwa_aerodyn.c
			sqrt					
			indicated_airspeed					rwa_aerodync.
			air_density					
			norm_vel					rwa_kinemat.c
			sin_aoa					rwa_kinemat.c
			cos_aoa					rwa_kinemat.c
			sin_yaw					rwa_kinemat.c
			cos_yaw					rwa_kinemat.c
			velocity_to_body					rwa_aerodyn.c
			ang_vel					rwa_kinemat.c
			vehicle_angular_velocity					
			rotate_get_mat					
			gravity					rwa_kinemat.c
			g_force					rwa_aerodyn.c
			vertical_speed					rwa_aerodyn.c
			vec_dot_prod					
			velocity_pitch					rwa_kinemat.c
			asin					
			body_pitch					rwa_kinemat.c
			roll					rwa_aerodyn.c
			heading					rwa_kinemat.c
			acos					
		het_simul						
		hydra_simul						rwa_hydra.c
			missile_hydra_fly_rockets					rkt_hydra.c
				rkts_in_flight				rkt_hydra.c
				hydra_fly				rkt_hydra.c
				missile_hydra_fly				
				ball_table				rkt_hydra.c
				missile_m73_init				
				missile_flechette_init				
				printf				
				missile_hydra_fly:missile_hydra_stop				
				missile_m73_drop				sub_m73.c
					missile_util_comm_check_sub_mun			util_comm.c
						printf		
						missile_comm		util_comm.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
						veh_kinematics		sun_stubs.c
						kinematics_range_squared		
						network_ifire_init_burst		
						network_ifire_send_detonation		
						map_get_ammo_entry_from_network_type		sun_wayed.c
						impacts_queue_effect		
						network_send_vehicle_impact		
					scaled_rand			
					sqrt			
					traj_up			sub_m73.c
					zero_velocity			sub_m73.c
					missile_util_comm_release_sub_munition			util_comm.c
						printf		
						veh_kinematics		sun_stubs.c
						kinematics_range_squared		
						missile_comm		util_comm.c
						store_traj_chord		
						event_get_eventid		
						vec_copy		
						d2f_vec_copy		
						map_get_ammo_entry_from_network_type		sun_wayed.c
						network_send_projectile_fire_pkt		
						impacts_queue_effect		
					missile_m73_get_impact			sub_m73.c
						scaled_rand		
						sin		
						vec_scale		
						vec_add		
				missile_m73_impact				sub_m73.c
					scaled_rand			
					missile_util_comm_check_sub_mun			util_comm.c
						printf		
						missile_comm		util_comm.c
						veh_kinematics		sun_stubs.c
						kinematics_range_squared		
						network_ifire_init_burst		
						network_ifire_send_detonation		
						map_get_ammo_entry_from_network_type		sun_wayed.c
						impacts_queue_effect		

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
						network_send_vehicle_impact		
					missile_m73_drop			sub_m73.c
						missile_util_comm_check_sub_mun		util_comm.c
							printf	
							missile_comm	util_comm.c
							veh_kinematics	sun_stubs.c
							kinematics_range_squared	
							network_ifire_init_burst	
							network_ifire_send_detonation	
							map_get_ammo_entry_from_network_type	sun_wayed.c
							impacts_queue_effect	
							network_send_vehicle_impact	
						scaled_rand		
						sqrt		
						traj_up		sub_m73.c
						zero_velocity		sub_m73.c
						missile_util_comm_release_sub_munition		util_comm.c
							printf	
							veh_kinematics	sun_stubs.c
							kinematics_range_squared	
							missile_comm	util_comm.c
							store_traj_chord	
							event_get_eventid	
							vec_copy	
							d2f_vec_copy	
							map_get_ammo_entry_from_network_type	sun_wayed.c
							network_send_projectile_fire_pkt	
							impacts_queue_effect	
						missile_m73_get_impact		sub_m73.c
							scaled_rand	
							sin	
							vec_scale	
							vec_add	
					zero_velocity			sub_mt3.c
					missile_util_comm_release_sub_munition			util_comm.c
						printf		
						veh_kinematics		sun_stubs.c
						kinematics_range_squared		
						missile_comm		util_comm.c

RWA AIRNET CALL TREE STRUCTURE

```
1st  2nd  3rd  4th  5th  6th  7th  8th
                              store_traj_chord
                              event_get_eventid
                              vec_copy
                              d2f_vec_copy
                              map_get_ammo_entry_from_network_type          sun_wayed.c
                              network_send_projectile_fire_pkt
                              impacts_queue_effect
                         printf
                    flechette veh_list                                      rkt_hydra.c
                    missile_flechette_fly                                   sub_flech.c
                         flechette_speed_coeff                              sub_flech.c
                         missile_util_eval_poly                             util_eval.c
                         vec_scale
                         vec_add
                         null_VehicleID                                     rwa_hydra.c
                         missile_fuze_all_prox                              fuze_prox.c
                              missile_fuze_prox                             fuze_prox.c
                                   missile_fuze_invest_prox                 fuze_prox.c
                                        printf
                                        near_get_next_veh_near_point
                                        get_prox                            fuze_prox.c
                                             free_ptr                       fuze_prox.c
                                             prox_free                      fuze_prox.c
                                             malloc
                                        near_get_veh_if_still_near_point
                                        free_prox
                                             free_ptr                       fuze_prox.c
                                             printf
                                             prox_free                      fuze_prox.c
                                             free
                              missile_fuze_detonate_prox                    fuze_prox.c
                                   vec_scale
                                   printf
                                   free_prox
                                        free_ptr                            fuze_prox.c
                                        printf
                                        prox_free                           fuze_prox.c
                                        free
                                   f2d_vec_scale                            fuze_prox.c
```

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th		
							vec_sub		
							vec_dot_prod		
							vec_add		
							f2d_mat_transpose		fuze_prox.c
							missile_util_comm_fuze_detonate		util_comm.c
								missile_comm	util_comm.c
								printf	
								vec_mat_mul	
					missile_util_comm_check_sub_mun				util_comm.c
						printf			
						missile_comm			util_comm.c
						veh_kinematics			sun_stubs.c
						kinematics_range_squared			
						network_ifire_init_burst			
						network_ifire_send_detonation			
						map_get_ammo_entry_from_network_type			sun_wayed.c
						impacts_queue_effect			
						network_send_vehicle_impact			
					zero_vector				sub_flech.c
					missile_util_comm_release_sub_munition				util_comm.c
						printf			
						veh_kinematics			sun_stubs.c
						kinematics_range_squared			
						missile_comm			util_comm.c
						store_traj_chord			
						event_get_eventid			
						vec_copy			
						d2f_vec_copy			
						map_get_ammo_entry_from_network_type			sun_wayed.c
						network_send_projectile_fire_pkt			
						impacts_queue_effect			
					missile_fuze_detonate_prox				fuze_prox.c
						vec_scale			
						printf			
					free_prox				fuze_prox.c
						free_ptr			fuze_prox.c
						printf			
						prox_free			fuze_prox.c
						free			

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
					f2d_vec_scale			fuze_prox.c
					vec_sub			
					vec_dot_prod			
					vec_add			
					f2d_mat_transpose			fuze_prox.c
					missile_util_comm_fuze_detonate			util_comm.c
						missile_comm		util_comm.c
						printf		
						vec_mat_mul		
				missile_fuze_prox_stop				fuze_prox.c
					free_prox			fuze_prox.c
						free_ptr		fuze_prox.c
						printf		
						prox_free		fuze_prox.c
						free		
				network_ifire_send_indirect_fire				
				missile_hydra_purge_free_missiles				rkt_hydra.c
					rkts_in_flight			rkt_hydra.c
					hydra_fly			rkt_hydra.c
			pylons_set					rwa_hydra.c
			pylon_R					
			rotate_set_no_rotate					
			pylon_L					
			articulation					
			left_rocket_launch					rwa_hydra.c
			hydra_launch_rocket					
			right_rocket_launch					rwa_hydra.c
		Lrf Post						
		Lrf Err String						
		fprintf						
	RTC_REPAIR_SIMUL							
	repair_simul							rwa_stubs.c
	RTC_NET_SIMUL							
	net_simul							
	io_simul							
	veh_spec_stop							rwa_main.c
		idc_reset						
		sound_reset						rwa_sound.c
			dont_use_sound					rwa_sound.c

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
			sounds					rwa_mem.c
			fifo_enqueue					
			sound_error					rwa_sound.c
				fprintf				
				fflush				
			veh_sound_array					rwa_sound.c
		vision_break_all_blocks						rwa_vision.c
			clear_view_flags					
			get_cig2_present					
		get_cig2-present						
		vision_clear_tc_board						rwa_vision.c
			clear_view_flags					
			set_view_flags					
		Lrf Un Init						
		Lrf Err String						
		fprintf						
	hull_uninit							
	sound_reset							rwa_sound.c
		dont_use_sound						rwa_sound.c
		sounds						rwa_mem.c
		fifo_enqueue						
		sound_error						rwa_sound.c
			fprintf					
			fflush					
		veh_sound_array						rwa_sound.c
	cig_uninit							
	dtad_uninit							
	bbd_uninit							
	veh_spec_exit							rwa_main.c
		keyboard_exit_gracefully						
		rwa_config_exit_gracefully						
		vision_break_all_blocks						
		timers get_current_time						
		printf						
		timers_get_current_tick						sun_stubs.c
		timers_elapsed_milliseconds						
		network_print_statistics						
		net_handle						
		net_close						

RWA AIRNET CALL TREE STRUCTURE

1st	2nd	3rd	4th	5th	6th	7th	8th	
	mem_free_shared_memory							main.c
	reboot_on_shutdown							

Appendix B - Source code listing for rwa_aerodyn.c.

The following appendix contains the source code listing for rwa_aerodyn.c for convenience in document maintenance and understanding of the CSU.

APPENDIX B - rwa_aerodyn.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_aerodyn.c,v 1.6
1993/01/28 23:33:00 cm-adst Exp $ */
/*
 * $Log: rwa_aerodyn.c,v $
 * Revision 1.6  1993/01/28  23:33:00  cm-adst
 * P. DesMeueles's changes for spcr 31
 *
 * Revision 1.5  1992/12/21  22:14:41  cm-adst
 * R. Branson's flight changes.  These changes will become
 * BDS-D 1.1.1.  This change was turned over by C. Swanson.
 *
 * Revision 1.1  1992/10/07  19:00:23  cm-adst
 * Initial Version
 *
*/
static char RCS_ID[] = "$Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_aerodyn.c,v 1.6 1993/01/28 23:33:00
cm-adst Exp $";

/*****************************************************************************
 *
 * Revisions:
 *
 *     Version        Date       Author       Title                        SP/CR Number
 *     _______      ________   __________   _________________________   ____________
 *
 *     1.2          10/09/92   R. Branson   Data File Initiali-
 *                                            zation
 *     1.3          10/16/92   R. Branson   Data filenames changed
 *                                            to eight charachters
 *     1.4          10/30/92   R. Branson   Added pathname to data
 *                                            directory
 *     1.5          01/19/93   P.Desmeules  Increased the size of the       31
 *                                          fgets to make sure the
 *                                          whole line is read in.
 *     1.5          03/04/93   P.Desmeules  Fix the value of                85
 *                                          HOVER_AUG_PITCH_RESET_VALUE
 *
 *****************************************************************************/

/*****************************************************************************
 *
 *     SP/CR No.    Description of Modification
 *     _________    ________________________________________________________
 *
 *                  Hard coded defines changed to array elements.
 *                  Aerodyn data array added.
 *                  Aerodyn initialization data array added.
 *                  Aerodyn stealth data array added.
 *                  Aerodyn simple data array added.
 *                  Added file read for aerodyn data, aerodyn initiali-
 *                     zation data, aerodyn stealth data, and aerodyn
 *                     simple data to the "aerodyn_init" function.
 *
 *                  Added "/simnet/data/" to each data file pathname.
```

APPENDIX B - rwa_aerodyn.c

```
*
*****************************************************************************/

/****************************************************************
 *                                                              *
 * FILE:          rwa_aerodyn.c                                 *
 * AUTHOR:        James Chung                                   *
 * MAINTAINER:    James Chung                                   *
 * HISTORY:       4/19/89  james: Creation                      *
 *                8/02/90  carol: added simplified aero dynamics*
 *                                                              *
 *                                                              *
 * Copyright (c) 1989 BBN Systems and Technologies Corporation  *
 * All rights reserved.                                         *
 *                                                              *
 * Interim aerodynamics model for a generic rotary-wing aircraft*
 * with flight characteristics similar to that of a McDonnell   *
 * Douglas AH-64 Apache attack helicopter.                      *
 *                                                              *
 ****************************************************************/

#include "stdio.h"
#include "simstdio.h"
#include "math.h"
#include "sim_dfns.h"
#include "sim_types.h"
#include "sim_macros.h"
#include "libmatrix.h"
#include "libmath.h"

#include "rwa_engine.h"
#include "vehicle.h"
#include "aero_param.h"
#include "std_atm.h"
#include "ground.h"
#include "rwa_ground.h"
#include "parameters.h"
#include "rwa_kinemat.h"
#include "libmun.h"
#include "libhull.h"
#include "libkin.h"
#include "rwa_aerodyn.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

#define MOMENT_OF_INERTIA_X              aero_data[ 0]
#define MOMENT_OF_INERTIA_Y              aero_data[ 1]
#define MOMENT_OF_INERTIA_Z              aero_data[ 2]
```

APPENDIX B - rwa_aerodyn.c

```
#define AIRFRAME_MASS                         aero_data[ 3]
#define ORDINANCE_MASS                        aero_data[ 4]
#define GRAV_CONSTANT                         aero_data[ 5]
#define CG_AC_X                               aero_data[ 6]
#define CG_AC_Y                               aero_data[ 7]
#define CG_AC_Z                               aero_data[ 8]

#define VIRTUAL_WING_AREA                     aero_data[ 9]
#define VIRTUAL_WING_COP_AC_X                 aero_data[10]
#define VIRTUAL_WING_COP_AC_Y                 aero_data[11]
#define VIRTUAL_WING_COP_AC_Z                 aero_data[12]
#define WING_LIFT_COEFFICIENT_FIT_3           aero_data[13]
#define WING_LIFT_COEFFICIENT_FIT_2           aero_data[14]
#define WING_LIFT_COEFFICIENT_FIT_1           aero_data[15]
#define WING_LIFT_COEFFICIENT_FIT_0           aero_data[16]
#define WING_STALL_AOA                (deg_to_rad(aero_data[17]))

#define VSTAB_AREA                            aero_data[18]
#define VSTAB_COP_AC_X                        aero_data[19]
#define VSTAB_COP_AC_Y                        aero_data[20]
#define VSTAB_COP_AC_Z                        aero_data[21]
#define VSTAB_LIFT_COEFFICIENT_1              aero_data[22]
#define VSTAB_STALL_SSA               (deg_to_rad(aero_data[23]))

#define MAIN_ROTOR_COP_AC_X                   aero_data[24]
#define MAIN_ROTOR_COP_AC_Y                   aero_data[25]
#define MAIN_ROTOR_COP_AC_Z                   aero_data[26]
#define MAIN_ROTOR_MAX_THRUST                 aero_data[27]
#define MAIN_ROTOR_MAST_TILT          (deg_to_rad(aero_data[28]))
#define MAIN_ROTOR_MAX_LOAD_TORQUE            aero_data[29]
#define MAIN_ROTOR_MAX_PITCH_MOMENT           aero_data[30]
#define MAIN_ROTOR_MAX_ROLL_MOMENT            aero_data[31]
#define MAIN_ROTOR_TORQUE_COUPLING_GAIN       aero_data[32]
#define MAIN_ROTOR_GROUND_EFFECT_FACTOR       aero_data[33]

#define TAIL_ROTOR_COP_AC_X                   aero_data[34]
#define TAIL_ROTOR_COP_AC_Y                   aero_data[35]
#define TAIL_ROTOR_COP_AC_Z                   aero_data[36]
#define TAIL_ROTOR_MAX_THRUST                 aero_data[37]
#define TAIL_ROTOR_MAX_LOAD_TORQUE            aero_data[38]

#define P_DRAG_COEFF_CONST                    aero_data[39]
#define P_DRAG_TAS_BREAK                      aero_data[40]
#define P_DRAG_COEFF_BREAK                    aero_data[41]
#define P_DRAG_TAS_MAX                        aero_data[42]
#define P_DRAG_COEFF_MAX                      aero_data[43]
#define TOTAL_WETTED_SURFACE_AREA             aero_data[44]

#define ATT_DAMPING_MODE_SIMPLE TRUE
/****************************************************************************
 Hover hold changes:

if ATT_DAMPING_MODE_SIMPLE
      when slow moving ( airspeed<10 knots ) the max pitch is 5 degrees
```

APPENDIX B - rwa_aerodyn.c

```
          medium       ( 10<=airspeed<30 )     pitch is 10 degrees
          other        ( 30<=airspeed )        pitch is 15 degrees
else
      when airspeed >= 10 knots pitch is proportional to log(speed)
      otherwise pitch is +/- 5 degrees

                                  Paul J. Metzger    11-1-89
**************************************************************************/
static REAL MAX_ATT_CTL_ANGLE;
#define MAX_ATT_CTL_ANGLE_STOP               aero_data[45]
#define MAX_ATT_DAMPING_FACTOR               aero_data[46]
#define HOVER_SLOW_LIMIT                     aero_data[47]
#define HOVER_AUG_PITCH_RESET_VALUE          aero_data[48]
static int  hover_hold_turned_on;          /* transition mode, TRUE or FALSE */

#if ATT_DAMPING_MODE_SIMPLE
#define MAX_ATT_CTL_ANGLE_NORM            (deg_to_rad (aero_data[49]))
#define MAX_ATT_CTL_ANGLE_MED       (deg_to_rad (aero_data[50]))
#define MAX_ATT_CTL_ANGLE_SLOW            (deg_to_rad (aero_data[51]))
#define HOVER_MED_LIMIT                      aero_data[52]
#endif


#define ATT_CTL_PITCH_P_GAIN                 aero_data[53]
#define ATT_CTL_PITCH_I_GAIN                 aero_data[54]
#define ATT_CTL_ROLL_P_GAIN                  aero_data[55]
#define ATT_CTL_ROLL_I_GAIN                  aero_data[56]

#define HOVER_AUG_ROLL_P_GAIN                aero_data[57]
#define HOVER_AUG_ROLL_I_GAIN                aero_data[58]
#define HOVER_AUG_PITCH_P_GAIN               aero_data[59]
#define HOVER_AUG_PITCH_I_GAIN               aero_data[60]
#define HOVER_AUG_YAW_P_GAIN                 aero_data[61]
#define HOVER_AUG_YAW_I_GAIN                 aero_data[62]
#define HOVER_AUG_CLIMB_P_GAIN               aero_data[63]
#define HOVER_AUG_CLIMB_I_GAIN               aero_data[64]
#define MAX_STAB_AUG_PITCH_ROLL_CONTROL      aero_data[65]
#define MAX_STAB_AUG_YAW_CLIMB_CONTROL       aero_data[66]

#define ROLL_RATE_DAMPING_GAIN               aero_data[67]
#define PITCH_RATE_DAMPING_GAIN              aero_data[68]
#define YAW_RATE_DAMPING_GAIN                aero_data[69]
#define VERTICAL_RATE_DAMPING_GAIN           aero_data[70]
#define LATERAL_VELOCITY_DAMPING_GAIN        aero_data[71]

#define LIFT_COEFF_VIRTUAL_WING              aero_data[72]
#define OSWALD_EFFIC_FACTOR                  aero_data[73]
#define INDUCED_DRAG_COEFF                   aero_data[74]

/*
* SPCR 85 - fix the value of HOVER_AUG_PITCH_RESET (element 48) from
* .44 to .044
*/
static REAL aero_data[100] = {
      50000.000, 50000.000, 50000.000,   4881.000,   1591.000,
```

```
        9.8,        0.0,        0.0,        -0.100,      25.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        1.0,        0.0,       30.0,         3.0,         0.0,
       -9.1,        0.0,        5.0,        60.0,         0.0,
        0.0,        2.0,   123500.0,         2.5,     76476.0,
   100000.0,   100000.0,        0.5,         0.4,         0.0,
       -9.1,        0.0,     8909.1,      1684.8,         0.0,
       50.0,        0.02,     100.0,         0.06,       50.0,
        6.0,        4.5,        5.15,        0.044,      15.0,
       10.0,        6.0,       15.46,        2.5,         0.05,
        5.0,        0.05,       0.1,         0.001,       0.1,
        0.001,     10.0,        5.0,         1.0,         0.5,
        0.2,        0.05,  100000.0,    100000.0,    100000.0,
     2000.0,     1000.0,        0.6,         0.9,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0
    } ;

static REAL aero_init[20] = {
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0
    } ;

static REAL aero_simple[20] = {
   500000.0,        0.5,       48.0,         0.15,       10.0,
      100.0,   150000.0,        1.5,         0.7,         0.03,
   400000.0,      100.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0
    } ;

static REAL aero_stealth[20] = {
       80.0,       30.0,       10.0, 10000000000.0, 10000000000.0,
     5000.0,    25000.0,        0.03,        0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0,
        0.0,        0.0,        0.0,         0.0,         0.0
    } ;

static int  hover_hold_state;                 /* OFF or ON */

static REAL MAIN_ROTOR_MAST_TILT_SIN;
static REAL MAIN_ROTOR_MAST_TILT_COS;

static REAL altitude;                         /* m */
static REAL true_airspeed;                    /* m/sec */
static REAL last_airspeed = 0;                /* m/sec */
static REAL vertical_speed;                   /* m/sec */
static REAL roll;                             /* rad */
static REAL pitch;                            /* rad */
static REAL roll_rate;                        /* rad/sec */
static REAL pitch_rate;                       /* rad/sec */
```

APPENDIX B - rwa_aerodyn.c

```
static REAL g_force;
static REAL last_g_force;
static REAL yaw_rate;                       /* rad/sec */
static REAL pitch_damping;
static REAL roll_damping;
static REAL yaw_damping;
static REAL ambient_temperature;            /* deg R */
static REAL ambient_pressure;               /* N / m^2 */
static REAL ambient_density;                /* kg / m^3 */
static REAL dynamic_pressure;               /* N / m^2 */
static REAL main_rotor_thrust;                    /* N */
static REAL tail_rotor_thrust;                    /* N */
static REAL lift_virtual_wing;                    /* N */
static REAL lift_vstab;
static REAL lift_coefficient_virtual_wing;
static REAL lift_coefficient_vstab;
static REAL total_drag;
static REAL total_incompressible_drag_coefficient;
static REAL gross_weight;                   /* N */
static REAL vehicle_mass;                   /* kg */
static REAL angle_of_attack;                /* rad */
static REAL side_slip_angle;                /* rad */
static REAL main_rotor_load_torque;         /* N-m */
static REAL tail_rotor_load_torque;         /* N-m */
static REAL powertrain_percent_shaft_speed;     /* 0-1 */

static REAL cyclic_pitch;              /* -1 to 1 */     /* Flight controls */
static REAL cyclic_roll;               /* -1 to 1 */     /* Flight controls */
static REAL collective;                /*  0 to 1 */
static REAL pedal;                     /* -1 to 1 */
static REAL stab_aug_pitch;
static REAL stab_aug_roll;
static REAL stab_aug_yaw;
static REAL stab_aug_climb;
static REAL stab_aug_pitch_integrator;
static REAL stab_aug_roll_integrator;
static REAL stab_aug_yaw_integrator;
static REAL stab_aug_climb_integrator;
static REAL hover_aug_pitch_angle;
static REAL hover_aug_roll_angle;
static REAL hover_aug_pitch_integrator;
static REAL hover_aug_roll_integrator;
static REAL attitude_control_roll_integrator;
static REAL attitude_control_pitch_integrator;
static REAL attitude_control_roll_command;
static REAL attitude_control_pitch_command;
static REAL controller_cyclic_pitch;
static REAL controller_cyclic_roll;
static REAL controller_collective;
static REAL controller_tail_rotor;

static REAL *angular_velocity_vector;     /* kinematic state vectors */
static REAL *normalized_velocity_vector;
static REAL *velocity_vector;
static REAL *gravity_dir_vector;
```

APPENDIX B - rwa_aerodyn.c

```
static REAL p_drag_fit_coeff[9];    /* parasite drag fit coefficients */

static REAL oswald_efficiency_factor;
static REAL induced_drag_coefficient;
static REAL parasite_drag_coefficient;

static VECTOR loc_ac_main_rotor_cop;
static VECTOR loc_ac_tail_rotor_cop;
static VECTOR loc_ac_virtual_wing_cop;
static VECTOR loc_ac_vstab_cop;
static VECTOR loc_ac_cg;

static VECTOR lift_body_virtual_wing;           /* body [X Y Z] */
static VECTOR lift_body_vstab;
static VECTOR force_body_main_rotor;
static VECTOR force_body_tail_rotor;
static VECTOR force_body_damping;
static VECTOR drag_body;
static VECTOR gravity_force_body;
static VECTOR force_ground_effect;
static VECTOR force_body;                   /* sum of all forces */

static VECTOR moment_body_virtual_wing;         /* body [X Y Z] */
static VECTOR moment_body_vstab;
static VECTOR moment_body_main_rotor;
static VECTOR moment_body_torque_coupling;
static VECTOR moment_body_tail_rotor;
static VECTOR moment_body_cg;
static VECTOR moment_body_damping;
static VECTOR moment_body;

static VECTOR virtual_wing_force;         /* velocity [H D L] */
static VECTOR vstab_force;
static VECTOR drag_force;

static T_MAT_PTR velocity_to_body;        /* vel -> body xform */

static T_MATRIX inertia_matrix =
{ {50000.0, 0, 0},
  {0, 50000.0, 0},
  {0, 0, 50000.0}};

int funny_little_kludge = 1;/* default is logarithmic for complex model */
static int aerodyn_debug = 0;

static int selected_model = COMPLEX_MODEL;  /* default: James' model */
static int allow_takeoff = TRUE;     /* allow stealth model to take off */
static int level_view = TRUE;        /* unset any pitch */
static REAL ground_height = 2.8;

void aero_body_point_set_front_wheels( distance_from_hull )
REAL distance_from_hull;
{
    body_point[0].position[Z] = distance_from_hull;
```

```
    body_point[1].position[Z] = distance_from_hull;
    ground_height = (REAL)(((int)(-distance_from_hull * 10)) / 10.0);
    printf( "Front Wheels set %1.41f m. under Hull.\n",
           distance_from_hull);
}

void aero_body_point_set_rear_wheel( distance_from_hull )
REAL distance_from_hull;
{
    body_point[2].position[Z] = distance_from_hull;
    printf( "Rear Wheel set %1.41f m. under Hull.\n",
           distance_from_hull);
}

REAL aero_get_ground_height()
{
    return( ground_height );
}

void aerodyn_init()
{
    int i;

/*  DEFAULT DATA FOR rwa_aerodyn.c READ FROM FILE                  */
    int     j;
    float   data_tmp;
    char    descript[80];

    FILE    *fp;

      P(printf("$$$$ RWA AERODYN $$$$\n"););

        fp = fopen("/simnet/data/rwa_aero.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rwa_aero.d\n");
            exit();
            }

      rewind(fp);

      /*    Read array data    */
      j=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            aero_data[j] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("aero_data(%3d) is%11.3f %s\n", j, aero_data[j],
                  descript););
            ++j;
            }

      fclose(fp);
/*  END DEFAULT DATA FOR rwa_aerodyn.c READ FROM FILE              */

/*  DEFAULT INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE  */
```

```
    fp = fopen("/simnet/data/rw_ae_in.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/rw_ae_in.d\n");
          exit();
          }

    rewind(fp);

    /*    Read array data    */
    j=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          aero_init[j] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("aero_init(%3d) is%11.3f %s\n", j, aero_init[j],
                descript););
          ++j;
          }

    fclose(fp);
/*  END DEFAULT INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE    */

/*  DEFAULT SIMPLE INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE */
    fp = fopen("/simnet/data/rw_ae_sp.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/rw_ae_sp.d\n");
          exit();
          }

    rewind(fp);

    /*    Read array data    */
    j=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          aero_simple[j] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("aero_simple(%3d) is%11.3f %s\n", j, aero_simple[j],
                descript););
          ++j;
          }

    fclose(fp);
/*  END DEFAULT SIMPLE INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE*/

/*  DEFAULT STEALTH INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE    */
    fp = fopen("/simnet/data/rw_ae_sl.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/rw_ae_sl.d\n");
          exit();
    }

    rewind(fp);

    /*    Read array data    */
```

```
      j=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            aero_stealth[j] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("aero_stealth(%3d) is%11.3f %s\n", j, aero_stealth[j],
                  descript););
            ++j;
      }

      fclose(fp);
/*  END DEFAULT STEALTH INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE*/

    engine_init();
    cyclic_pitch =                    aero_init[ 0];
    cyclic_roll =               aero_init[ 1];
    if (selected_model != STEALTH_MODEL)
        collective =                  aero_init[ 2];
    else
    {
        collective = 0.5;
      allow_takeoff = TRUE;
    }
    pedal =                           aero_init[ 3];

    stab_aug_pitch_integrator =       aero_init[ 4];
    stab_aug_roll_integrator =              aero_init[ 5];
    stab_aug_yaw_integrator =               aero_init[ 6];
    stab_aug_climb_integrator =       aero_init[ 7];
    attitude_control_pitch_integrator = aero_init[ 8];
    attitude_control_roll_integrator =    aero_init[ 9];
    hover_aug_pitch_integrator =      aero_init[10];
    hover_aug_roll_integrator =       aero_init[11];
    hover_aug_pitch_angle =           aero_init[12];
    hover_aug_roll_angle =            aero_init[13];

    hover_hold_state = OFF;
    hover_hold_turned_on = FALSE;

    loc_ac_main_rotor_cop[X] = MAIN_ROTOR_COP_AC_X;
    loc_ac_main_rotor_cop[Y] = MAIN_ROTOR_COP_AC_Y;
    loc_ac_main_rotor_cop[Z] = MAIN_ROTOR_COP_AC_Z;

    loc_ac_tail_rotor_cop[X] = TAIL_ROTOR_COP_AC_X;
    loc_ac_tail_rotor_cop[Y] = TAIL_ROTOR_COP_AC_Y;
    loc_ac_tail_rotor_cop[Z] = TAIL_ROTOR_COP_AC_Z;

    loc_ac_virtual_wing_cop[X] = VIRTUAL_WING_COP_AC_X;
    loc_ac_virtual_wing_cop[Y] = VIRTUAL_WING_COP_AC_Y;
    loc_ac_virtual_wing_cop[Z] = VIRTUAL_WING_COP_AC_Z;

    loc_ac_vstab_cop[X] = VSTAB_COP_AC_X;
    loc_ac_vstab_cop[Y] = VSTAB_COP_AC_Y;
    loc_ac_vstab_cop[Z] = VSTAB_COP_AC_Z;
```

```
    loc_ac_cg[X] = CG_AC_X;
    loc_ac_cg[Y] = CG_AC_Y;
    loc_ac_cg[Z] = CG_AC_Z;

    inertia_matrix[1] [1] = MOMENT_OF_INERTIA_X;
    inertia_matrix[2] [2] = MOMENT_OF_INERTIA_Y;
    inertia_matrix[3] [3] = MOMENT_OF_INERTIA_Z;

    pitch_damping = PITCH_RATE_DAMPING_GAIN;
    roll_damping = ROLL_RATE_DAMPING_GAIN;
    yaw_damping = YAW_RATE_DAMPING_GAIN;

    MAIN_ROTOR_MAST_TILT_SIN = sin(MAIN_ROTOR_MAST_TILT);
    MAIN_ROTOR_MAST_TILT_COS = cos(MAIN_ROTOR_MAST_TILT);

    vec_init (vstab_force);
    vec_init (drag_force);
    vec_init (ground_force);
    vec_init (force_ground_effect);
    vec_init (force_body);
    vec_init (moment_body);
    vec_init (moment_body_torque_coupling);
    vec_init (force_body_main_rotor);
    vec_init (force_body_tail_rotor);
    vec_init (force_body_damping);

    vehicle_mass_init (AIRFRAME_MASS + ORDINANCE_MASS, inertia_matrix );
    ground_init();

    for (i=0; i<9; i++)              /* Set parasite drag profile */
    {
      p_drag_fit_coeff[i] = 0.0;
    }

    if (find_cubic_func (0.0, P_DRAG_COEFF_CONST,
                  P_DRAG_TAS_BREAK, P_DRAG_COEFF_BREAK,
                  P_DRAG_TAS_MAX, P_DRAG_COEFF_MAX,
                  0.5, p_drag_fit_coeff) != TRUE)
    {
      fprintf (stderr, "AERODYN: Error - unable to fit p_drag function\n");
    }

/* So one can tweak the constants without recompiling */

    if (selected_model)
        aerodyn_read_simple_constants (get_constants_file ());

}

static void get_aircraft_kinematic_state()
{

    orientation_calc();
    parameters_calc();
```

APPENDIX B - rwa_aerodyn.c

```
    true_airspeed = kinematics_get_true_airspeed();
    altitude = kinematics_get_altitude();
    angular_velocity_vector = kinematics_get_angular_velocity_vector();
    normalized_velocity_vector = kinematics_get_normalized_velocity_vector();
    velocity_vector = kinematics_get_linear_velocity_vector();
    gravity_dir_vector = kinematics_get_gravity_vector();
    angle_of_attack = kinematics_get_aoa();
    side_slip_angle = - kinematics_get_yaw();
    velocity_to_body = kinematics_get_velocity_to_body();
    g_force =  kinematics_get_g_force();
    vertical_speed = kinematics_get_vertical_speed();
}

static void deb_mat_print (m)
    T_MATRIX m;
{
    int i;
    for (i=0; i<=2; i++)
    {
        printf("%0.3lf   %0.3lf   %0.3lf\n", m[i][0], m[i][1], m[i][2]);
    }
}

static void compute_flight_parameters()
{
    ambient_density = air_density(altitude);
    ambient_temperature = air_temperature(altitude);
    ambient_pressure = air_pressure(altitude);
    dynamic_pressure = 0.5 * ambient_density * square (true_airspeed);
    pitch_rate = angular_velocity_vector[X];
    roll_rate = angular_velocity_vector[Y];
    yaw_rate = angular_velocity_vector[Z];
    roll = atan2 (-gravity_dir_vector[X], -gravity_dir_vector[Z]);
    pitch = atan2 (-gravity_dir_vector[Y], -gravity_dir_vector[Z]);
}

static void interact_with_ground()
{
    REAL brake_factor;

    brake_factor = normalized_velocity_vector[Y] *
                   true_airspeed / (true_airspeed + 5);
    body_point[0].x_force = - 6000 * brake_factor;
    body_point[1].x_force = body_point[0].x_force;

    ground_interaction(ground_force,ground_torque,body_point,grnd,
            NUMBER_OF_BODY_POINTS);

    force_ground_effect[Z] = main_rotor_thrust
                   * MAIN_ROTOR_GROUND_EFFECT_FACTOR
                   / (cig_altitude_above_gnd() + 1.0);
}

/*********************************************/
/* fuel get current level returns gallons      */
```

APPENDIX B - rwa_aerodyn.c

```
/*   gals * (6.5 lbs / gal) * (1kg / 2.2 lbs)    */
/*********************************************/
#define KILOGRAMS_PER_GALLON  2.95454545454

static void compute_gross_weight()
{
    vehicle_mass = AIRFRAME_MASS + ORDINANCE_MASS +
      fuel_get_current_level() * KILOGRAMS_PER_GALLON;/* kg */

    gross_weight = vehicle_mass * GRAV_CONSTANT;      /* N */
}

void aerodyn_set_lateral_stick (val)
    REAL val;
{
    cyclic_roll = -val;
}

void aerodyn_set_longitudinal_stick (val)
    REAL val;
{
    cyclic_pitch = -val;
}

void aerodyn_set_pedal (val)
    REAL val;
{
    pedal = val;
}

void aerodyn_set_collective (val)
    REAL val;
{
    if (funny_little_kludge)
      collective = log10 (val * 9.0 + 1.0); /* or, how to make linear log */
    else
        collective = val;
}

static void compute_lift_drag_forces()
{
    lift_virtual_wing = dynamic_pressure *
            lift_coefficient_virtual_wing * VIRTUAL_WING_AREA;

    lift_vstab = dynamic_pressure * lift_coefficient_vstab * VSTAB_AREA;

    total_drag = total_incompressible_drag_coefficient * dynamic_pressure *
                  TOTAL_WETTED_SURFACE_AREA;
}

static void compute_body_damping_forces_and_moments()
{
    moment_body_damping[X] = - pitch_damping * pitch_rate;
    moment_body_damping[Y] = - roll_damping * roll_rate;
    moment_body_damping[Z] = - yaw_damping * yaw_rate;
```

APPENDIX B - rwa_aerodyn.c

```
    force_body_damping[X] = -velocity_vector[X] * LATERAL_VELOCITY_DAMPING_GAIN;
    force_body_damping[Y] = 0.0;
    force_body_damping[Z] = -velocity_vector[Z] * VERTICAL_RATE_DAMPING_GAIN;
}

static REAL virtual_wing_lift_coefficient (alpha)
    REAL alpha;
{
    if (alpha > WING_STALL_AOA || alpha < 0.0)
      return (0.0);
    else
    return    (((WING_LIFT_COEFFICIENT_FIT_3 *  alpha +
               WING_LIFT_COEFFICIENT_FIT_2) * alpha +
               WING_LIFT_COEFFICIENT_FIT_1) * alpha +
               WING_LIFT_COEFFICIENT_FIT_0);
}

static REAL vstab_lift_coefficient (yaw)
    REAL yaw;
{
    REAL yawval;

    if (abs(yaw) > VSTAB_STALL_SSA)
      yawval = sign(yawval) * VSTAB_STALL_SSA;
    else
      yawval = yaw;

    return (VSTAB_LIFT_COEFFICIENT_1 * yawval);
}

static void compute_lift_drag_coefficients()
{
    REAL multiplier;

    lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
/* Computing virtual wing coefficient as independent of AOA */
    lift_coefficient_virtual_wing = LIFT_COEFF_VIRTUAL_WING;
/*                 virtual_wing_lift_coefficient (angle_of_attack); */

    parasite_drag_coefficient = cubic_func (true_airspeed, p_drag_fit_coeff);

    if (true_airspeed > 0.0 && angle_of_attack > 0.0) /* speed brake */
    {
      multiplier = 5.0 * true_airspeed * sin(angle_of_attack);
      if (multiplier > 1.0)
          parasite_drag_coefficient *= multiplier;
    }

    oswald_efficiency_factor = OSWALD_EFFIC_FACTOR;

    induced_drag_coefficient = INDUCED_DRAG_COEFF;

    total_incompressible_drag_coefficient = parasite_drag_coefficient +
                                    induced_drag_coefficient;
```

```
}

static void send_to_dynamics_kinematics()
{
    vehicle_mass_init (vehicle_mass, inertia_matrix);
    vehicle_forces (force_body);
    vehicle_torques (moment_body);
}

void dump_forces()
{
    vec_dump ("lift_body_virtual_wing:", lift_body_virtual_wing);
    vec_dump ("lift_body_vstab:", lift_body_vstab);
    vec_dump ("drag_body:", drag_body);
    vec_dump ("gravity_force_body:", gravity_force_body);
    vec_dump ("force_body_main_rotor:", force_body_main_rotor);
    vec_dump ("force_body_tail_rotor:", force_body_tail_rotor);
    vec_dump ("ground_force:", ground_force);
    vec_dump ("force_body:", force_body);
}

static void sum_body_forces_and_moments_about_ac()
{
    vec_init (force_body);
    vec_add (force_body, force_body_main_rotor, force_body);
/*    vec_add (force_body, force_body_tail_rotor, force_body); */
    vec_add (force_body, lift_body_virtual_wing, force_body);
    vec_add (force_body, lift_body_vstab, force_body);
    vec_add (force_body, drag_body, force_body);
    vec_add (force_body, force_body_damping, force_body);
    vec_add (force_body, gravity_force_body, force_body);
    vec_add (force_body, ground_force,force_body);
    vec_add (force_body, force_ground_effect, force_body);

    vec_cross_prod(loc_ac_tail_rotor_cop, force_body_tail_rotor,
                        moment_body_tail_rotor);
    vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                        moment_body_virtual_wing);
    vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
    vec_cross_prod(loc_ac_cg, gravity_force_body, moment_body_cg);

    vec_init (moment_body);
    vec_add (moment_body, moment_body_main_rotor, moment_body);
    vec_add (moment_body, moment_body_tail_rotor, moment_body);
    vec_add (moment_body, moment_body_virtual_wing, moment_body);
    vec_add (moment_body, moment_body_vstab, moment_body);
    vec_add (moment_body, moment_body_cg, moment_body);
    vec_add (moment_body, ground_torque, moment_body);
    vec_add (moment_body, moment_body_damping, moment_body);
}


static void transform_lift_drag_forces_to_body_coordinates()
{
    virtual_wing_force[Z]  = lift_virtual_wing; /* [H, D, L] */
```

```
    vstab_force[X] =  lift_vstab;
    drag_force[Y] = -total_drag;

    if (true_airspeed < P_DRAG_TAS_BREAK)       /* jwc 8/90 */
      drag_force[Y] -= sin(pitch) * 50000;

    vec_mat_mul (virtual_wing_force, velocity_to_body, lift_body_virtual_wing);
    vec_mat_mul (vstab_force, velocity_to_body, lift_body_vstab);
    vec_mat_mul (drag_force, velocity_to_body, drag_body);
}

static void generate_gravity_body_force()
{
    compute_gross_weight();

    gravity_force_body[X] = gravity_dir_vector[X] * gross_weight;
    gravity_force_body[Y] = gravity_dir_vector[Y] * gross_weight;
    gravity_force_body[Z] = gravity_dir_vector[Z] * gross_weight;
}

static int frame;

void aerodyn_debug_print()
{
    REAL roll, pitch, yaw, heading, airspeed_knots, weight_lbs, thrust_lbs;
    REAL *position;
    roll=atan2(-gravity_dir_vector[X],-gravity_dir_vector[Z]) *180.0 / 3.1416;
    pitch=atan2(-gravity_dir_vector[Y],-gravity_dir_vector[Z])*180.0 / 3.1416;
    yaw = side_slip_angle;
    airspeed_knots = true_airspeed * 3.26 / 1.69;
    weight_lbs = gross_weight / 9.8 * 2.2;
    position = vehicle_A_p();
    heading = rad_to_deg (kinematics_get_heading());
    printf ("KTAS = %0.21f VV = %0.31f  %0.31f  %0.31f YR = %0.31f\n",
      airspeed_knots, velocity_vector[X], velocity_vector[Y],
      velocity_vector[Z], angular_velocity_vector[Z]);
    printf ("xyzh = %0.31f %0.31f %0.31f %0.21f rpy = %0.31f %0.31f %0.31f\n",
      position[X], position[Y], position[Z], heading,
      roll, pitch, yaw);
    if (hover_hold_state == ON)
        printf ("stab_aug[rpyc]: %0.31f  %0.31f  %0.31f  %0.31f\n",
      stab_aug_roll, stab_aug_pitch, stab_aug_yaw, stab_aug_climb);
}

static void compute_rotor_loads()
{
    main_rotor_load_torque = controller_collective *
                            MAIN_ROTOR_MAX_LOAD_TORQUE;
    tail_rotor_load_torque = abs (controller_tail_rotor) *
                            TAIL_ROTOR_MAX_LOAD_TORQUE;
}

static void compute_engine_torque()
{
    engine_simul(main_rotor_load_torque, tail_rotor_load_torque, altitude);
```

APPENDIX B - rwa_aerodyn.c

```
    powertrain_percent_shaft_speed = engine_get_rotor_percent_shaft_speed();
}

static void compute_rotor_forces_and_moments()
{
    main_rotor_thrust = powertrain_percent_shaft_speed * controller_collective
            * MAIN_ROTOR_MAX_THRUST;

    tail_rotor_thrust = powertrain_percent_shaft_speed * controller_tail_rotor
            * TAIL_ROTOR_MAX_THRUST;

    force_body_main_rotor[Y] = main_rotor_thrust * MAIN_ROTOR_MAST_TILT_SIN;

    force_body_main_rotor[Z] = main_rotor_thrust * MAIN_ROTOR_MAST_TILT_COS;

    force_body_tail_rotor[X] = tail_rotor_thrust;

    moment_body_main_rotor[X] =
            - controller_cyclic_pitch * MAIN_ROTOR_MAX_PITCH_MOMENT;
    moment_body_main_rotor[Y] =
            controller_cyclic_roll * MAIN_ROTOR_MAX_ROLL_MOMENT;
    moment_body_main_rotor[Z] =
            - main_rotor_load_torque * MAIN_ROTOR_TORQUE_COUPLING_GAIN;
}
static REAL limiter (lower, val, upper)
REAL lower, val, upper;
{
    if (val > upper) return (upper);
    else if (val < lower) return (lower);
    else return (val);
}

static REAL set_roll_attitude (angle)
REAL angle;
{
    attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN * (roll - angle);
    /**** These used to be attitude_control_pitch_integrator instead of
        attitude_control_roll_integrator.          PJM    11-1-89
    attitude_control_pitch_integrator =
            limiter (-0.1, attitude_control_pitch_integrator, 0.1);
    ******/
    attitude_control_roll_integrator =
            limiter (-0.1, attitude_control_roll_integrator, 0.1);
    attitude_control_roll_command = ATT_CTL_ROLL_P_GAIN * (roll - angle);
    attitude_control_roll_command += attitude_control_roll_integrator;
    attitude_control_roll_command =   limiter (-MAX_STAB_AUG_PITCH_ROLL_CONTROL,
                                  attitude_control_roll_command,
                                  MAX_STAB_AUG_PITCH_ROLL_CONTROL);
    return (attitude_control_roll_command);
}

static REAL set_pitch_attitude (angle)
REAL angle;
{
    attitude_control_pitch_integrator +=
```

```
            ATT_CTL_PITCH_I_GAIN * (pitch - angle);
    attitude_control_pitch_integrator =
            limiter (-0.1, attitude_control_pitch_integrator, 0.1);
    attitude_control_pitch_command = ATT_CTL_PITCH_P_GAIN * (pitch - angle);
    attitude_control_pitch_command += attitude_control_pitch_integrator;
    attitude_control_pitch_command =  limiter (-MAX_STAB_AUG_PITCH_ROLL_CONTROL,
                                    attitude_control_pitch_command,
                                    MAX_STAB_AUG_PITCH_ROLL_CONTROL);
    return (attitude_control_pitch_command);
}

static void compute_stab_augmentation_gains()
{
    if (hover_hold_state == ON)
    {
      if ( !hover_hold_turned_on )
            {
            hover_hold_turned_on = TRUE ;

            pitch_damping = 2 * PITCH_RATE_DAMPING_GAIN;  /* jwc 8/90 */
            roll_damping = 2 * ROLL_RATE_DAMPING_GAIN;

            /* You should already be "hovering" (airspeed < 10 knots)
               for hover hold to show little visible swaying. */

            hover_aug_roll_integrator = 0.0 ;
            hover_aug_pitch_integrator = HOVER_AUG_PITCH_RESET_VALUE ;
            stab_aug_yaw_integrator = 0.0 ;
            stab_aug_climb_integrator = 0.0 ;

#if ATT_DAMPING_MODE_SIMPLE
            if  (true_airspeed < HOVER_SLOW_LIMIT)
                {
                if (true_airspeed > -HOVER_SLOW_LIMIT)
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_SLOW ;
                else if (true_airspeed > -HOVER_MED_LIMIT)
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_MED;
                else
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_NORM ;
                }
            else if (true_airspeed < HOVER_MED_LIMIT)
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_MED ;
            else
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_NORM ;
#endif
            }


#if ATT_DAMPING_MODE_SIMPLE
      if (true_airspeed > HOVER_SLOW_LIMIT )
            MAX_ATT_CTL_ANGLE =
                  log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
      else if (true_airspeed < -HOVER_SLOW_LIMIT )
            MAX_ATT_CTL_ANGLE =
                  log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
```

```
      else
            MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_STOP ;

      MAX_ATT_CTL_ANGLE = deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif


      hover_aug_roll_integrator +=
                  HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
      hover_aug_roll_integrator =
                  limiter(-0.2,hover_aug_roll_integrator,0.2);
      hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN * velocity_vector[X]
                  + hover_aug_roll_integrator;
      hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_roll_angle,
                              MAX_ATT_CTL_ANGLE);
        stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);

      hover_aug_pitch_integrator +=
                  HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
      hover_aug_pitch_integrator =
                  limiter(-0.2,hover_aug_pitch_integrator,0.2);
      hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN * velocity_vector[Y]
                  + hover_aug_pitch_integrator;
      hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_pitch_angle,
                              MAX_ATT_CTL_ANGLE);
      stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);

      stab_aug_yaw_integrator -=
                  HOVER_AUG_YAW_I_GAIN * angular_velocity_vector[Z];
      if (stab_aug_yaw_integrator >  0.5) stab_aug_yaw_integrator = 0.5;
      if (stab_aug_yaw_integrator < -0.5) stab_aug_yaw_integrator = -0.5;
      stab_aug_yaw    = - HOVER_AUG_YAW_P_GAIN * angular_velocity_vector[Z] +
                        stab_aug_yaw_integrator;

      stab_aug_climb_integrator -=
                  HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
      if (stab_aug_climb_integrator >  0.2) stab_aug_climb_integrator = 0.2;
      if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
      stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN * velocity_vector[Z] +
                        stab_aug_climb_integrator;

      stab_aug_yaw = limiter (-MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                        stab_aug_yaw,
                        MAX_STAB_AUG_YAW_CLIMB_CONTROL);

      stab_aug_climb = limiter (-MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                        stab_aug_climb,
                        MAX_STAB_AUG_YAW_CLIMB_CONTROL);
    }
    else
    {
      stab_aug_roll = 0.0;
      stab_aug_pitch = 0.0;
```

```
      stab_aug_yaw = 0.0;
      stab_aug_climb = 0.0;

      pitch_damping = PITCH_RATE_DAMPING_GAIN;  /* jwc 8/90 */
      roll_damping =  ROLL_RATE_DAMPING_GAIN;

#ifdef notdef
      hover_aug_roll_integrator = 0.0;    /* added 8/31/89 (jwc) */
      hover_aug_pitch_integrator = 0.0;
#endif
    }
    controller_cyclic_roll = cyclic_roll + stab_aug_roll;
    controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
    controller_tail_rotor = pedal + stab_aug_yaw;
    controller_collective = collective + stab_aug_climb;
}

static void send_aero_data_to_displays()
{
    if (velocity_vector[Y] > 0.0)
        meter_air_speed_set(true_airspeed);
    else
      meter_air_speed_set (0.0);

    meter_altitude_set(altitude);
    meter_vertical_speed_set(vertical_speed);
}

void aerodyn_simul()
{
    get_aircraft_kinematic_state();
    compute_flight_parameters();
    compute_stab_augmentation_gains();
    compute_rotor_loads();
    compute_engine_torque();
    compute_rotor_forces_and_moments();
    compute_lift_drag_coefficients();
    compute_lift_drag_forces();
    compute_body_damping_forces_and_moments();
    transform_lift_drag_forces_to_body_coordinates();
    generate_gravity_body_force();
    interact_with_ground();
    sum_body_forces_and_moments_about_ac();
    send_to_dynamics_kinematics();
/*  send_aero_data_to_displays();  Must call if not calling orientation_calc */
    vehicle_update();
}

REAL aerodyn_get_true_airspeed()
{
    return (true_airspeed);
}

void aerodyn_set_hover_hold_on ()
{
```

```
    hover_hold_state = ON;
}

void aerodyn_set_hover_hold_off()
{
    hover_hold_state = OFF;
    hover_hold_turned_on = FALSE;
    level_view = TRUE;
}

void aerodyn_toggle_hover_hold()
{
    if (hover_hold_state == OFF)
      hover_hold_state = ON;
    else
      {
      hover_hold_state = OFF;
      hover_hold_turned_on = FALSE;
      }
}

void forces_init ()
{
    aerodyn_init();
}


/*****************************************************************************
 * The following stuff is for the simplified dynamics model.  The model is *
 * a modification of the aerodynamics model Warren wrote for the SAF.      *
 * Global variables defined for the real aerodynamics are reused here to   *
 * allow overlap in generic routines for operations such as control inputs,*
 * init, etc.  - CJC                                                        *
 *****************************************************************************/


#define MAX_HELICOPTER_POWER  aero_simple[ 0]
#define MAX_HH                aero_simple[ 1]

/* constants for tweaking */
#define H_K1                  aero_simple[ 2]
#define H_K2                  aero_simple[ 3]

/* as increase drag coefficients, helicopter slows down faster */
#define H_K7                  aero_simple[ 4]
#define H_K8                  aero_simple[ 5]
#define H_KP                  aero_simple[ 6]
#define H_KPR                 aero_simple[ 7]
#define H_KY                  aero_simple[ 8]
#define H_KH                  aero_simple[ 9]
#define H_CHH                 aero_simple[10]
#define H_CL                  aero_simple[11]

void aerodyn_simple_simul ()
{
```

```
register int i;
register REAL *vec_ptr;
register REAL *res_ptr;
register REAL *cur_ptr;
register REAL *des_ptr;
REAL *drag_ptr;
REAL power;
REAL coll_factor;
REAL lift_factor;

VECTOR orient_vec;
VECTOR angular_accel;
VECTOR hover_hold_additions;
REAL euler[3];       /* euler angles */
VECTOR gravity_vector;    /* in body coordinates   */
T_MAT_PTR C_mat;     /* direction cosine matrix */

get_aircraft_kinematic_state ();
generate_gravity_body_force();
compute_rotor_loads();
compute_engine_torque();

if (hover_hold_state == ON)
{
  hover_hold_additions[0] = min(velocity_vector[1] * H_KH,MAX_HH);
  hover_hold_additions[0] = max(hover_hold_additions[0],-MAX_HH);
  hover_hold_additions[1] = min(- velocity_vector[0] * H_KH,MAX_HH);
  hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
  hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
}
else
{
  hover_hold_additions[0] = 0;
  hover_hold_additions[1] = 0;
  hover_hold_additions[2] = 0;
}

lift_factor = velocity_vector[1] * velocity_vector[1] * H_CL *
                   - cyclic_pitch;

/** original comment from SAF code **/
/* may want to put in power limit per unit time ... */
coll_factor = max(0.0,collective - 0.3);
power = H_KP * coll_factor + hover_hold_additions[2];
power +=  gross_weight * collective/(H_K2+collective) * 1.25;
power = min (MAX_HELICOPTER_POWER, power);
power = max (0.0, power);

if (fuel_level_empty ())
    power = 0.0;

/* Calculate the torque required to achieve the desired orientation */
/* orientation vector is [pitch element, roll element, yaw element] */

orient_vec[0] = H_KPR * - cyclic_pitch + hover_hold_additions[0];
```

```
    orient_vec[1] = H_KPR * cyclic_roll + hover_hold_additions[1];

    /** yaw element = current_yaw (heading) + rudder (pedals) * K **/
    orient_vec[2] = kinematics_get_yaw () + sign(pedal) * pedal
                      * pedal * H_KY;

    res_ptr = moment_body;
    des_ptr = orient_vec;

    C_mat = kinematics_get_w_to_h (veh_kinematics);
    euler[0] = atan2 (-gravity_dir_vector[Y], -gravity_dir_vector[Z]);
    euler[1] = - atan2 (-gravity_dir_vector[X], -gravity_dir_vector[Z]);
    euler[2] = kinematics_get_yaw ();
    cur_ptr = euler;

    /*  First, compute the angular velocity necessary to achieve the */
    /*  desired orientation in exactly one tick.  (delta theta/ delta T) */
    /*  Then get the angular acceleration needed to get to that velocity */
    /*  In one tick.*/
    for (i = X; i <= Z; ++i)
    {
      vec_ptr[i] = ((des_ptr[i] - cur_ptr[i]) / DELTA_T / H_K1);
      angular_accel[i] = (vec_ptr[i] - angular_velocity_vector[i])
                          / DELTA_T;
      res_ptr[i] = MOMENT_OF_INERTIA_X * angular_accel[i];
    }
    res_ptr[X] += lift_factor;    /* this should add some torque for turns */

    /* compute force vector */
    res_ptr = force_body;
    cur_ptr = velocity_vector;
    vec_ptr = euler;
    drag_ptr = drag_force; /* drag_body or drag_force */

    drag_ptr[X] = square(cur_ptr[X]) * H_K8;
    drag_ptr[Y] = square(cur_ptr[Y]) * H_K7;
    drag_ptr[Z] = square(cur_ptr[Z]) * H_K8;

    res_ptr[X] = (sin(vec_ptr[Y]) * power) - (sign(cur_ptr[X]) * drag_ptr[X]);
    res_ptr[Y] = -(sin(vec_ptr[X]) * power) - (sign(cur_ptr[Y]) * drag_ptr[Y]);

    res_ptr[Z] = C_mat[2][2] * power;
    res_ptr[Z] -= sign(cur_ptr[Z]) * drag_ptr[Z];
    res_ptr[Z] += lift_factor;    /* this should add some force for lift */

    vec_add (force_body, ground_force,force_body);
    vec_add (force_body, gravity_force_body,force_body);
    interact_with_ground();
    vec_add (force_body, force_ground_effect, force_body);
    vec_add (moment_body, ground_torque, moment_body);
    send_to_dynamics_kinematics ();
    vehicle_update ();
}

/***********************************************************************
```

APPENDIX B - rwa_aerodyn.c

```
 * The following is for the simplified model incorporating the stealth *
 * dynamics. In this model, the cyclic changes the desired velocity     *
 ***********************************************************************/

#define H_FWD_MUL aero_stealth[ 0]
#define H_SIDE_MUL      aero_stealth[ 1]
#define H_COLL_MUL      aero_stealth[ 2]
#define MAX_TORQUE      aero_stealth[ 3]
#define MAX_FORCE aero_stealth[ 4]
#define MASS            aero_stealth[ 5]
#define INERTIA         aero_stealth[ 6]
#define DEAD_ZONE aero_stealth[ 7]

/* use for gravity frame matrix.  eliminate all pitch and roll
 * start with identity.  substitute cos (yaw) for last term.
 */

static T_MATRIX level = {{1.0, 0.0, 0.0},
                         {0.0, 1.0, 0.0},
                         {0.0, 0.0, 1.0}};


void aerodyn_stealth_simul ()
{
    VECTOR desired_rot_vel;
    VECTOR desired_lin_vel;
    REAL adj_collective; /* collective value adjusted for dead zone and
                            for -1 to 1 range */

    adj_collective = (collective - 0.5) * 2.0;   /* change to -1 to 1 */

    if (aerodyn_debug)
       timed_printf ("adj_collective = %.3lf\n", adj_collective);

    if (allow_takeoff)
    {
      if (adj_collective > 0.0)
      {
          allow_takeoff = FALSE;
      }
      else
      {
          adj_collective = 0.0;
      }
    }

    get_aircraft_kinematic_state ();
    compute_rotor_loads();
    compute_engine_torque();

 /* update desired velocity */
    desired_lin_vel[Z] = adj_collective * adj_collective *
          sign (adj_collective ) * H_COLL_MUL;

    if (hover_hold_state == ON)
```

```
    {  /* no linear velocity in X,Y, only pitch */
      desired_lin_vel[X] = desired_lin_vel[Y] = 0.0;
      desired_rot_vel[X] = -cyclic_pitch * cyclic_pitch * sign(cyclic_pitch);
      desired_rot_vel[Y] = 0.0;
    }
    else
    {
      if (level_view)/* when not in pitch mode, level view */
      {
            vehicle_set_orientation_matrix (level); /* identity matrix */
            vehicle_set_orientation (kinematics_get_heading());
          level_view = FALSE;
      }

        desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
              * H_SIDE_MUL;
        desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
              * H_FWD_MUL;

        desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
    }
#ifdef notdef
        desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
              * H_SIDE_MUL;
        desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
              * H_FWD_MUL;

        desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
#endif
    desired_rot_vel[Z] =  pedal * pedal * sign(pedal);

    /* controller_forces */

    force_body[X] = (desired_lin_vel[X] - velocity_vector[X])
                    * MASS/DELTA_T;
    force_body[Y] = (desired_lin_vel[Y] - velocity_vector[Y])
                    * MASS/DELTA_T;
    force_body[Z] = (desired_lin_vel[Z] - velocity_vector[Z])
                    * MASS/DELTA_T;
    force_body[X] = min (MAX_FORCE, force_body[X]);
    force_body[Y] = min (MAX_FORCE, force_body[Y]);
    force_body[Z] = min (MAX_FORCE, force_body[Z]);

    force_body[X] = max (-MAX_FORCE, force_body[X]);
    force_body[Y] = max (-MAX_FORCE, force_body[Y]);
    force_body[Z] = max (-MAX_FORCE, force_body[Z]);


    /* controller_torques */
    moment_body[X] = (desired_rot_vel[X] - angular_velocity_vector[X])
                     * INERTIA/DELTA_T;
    moment_body[Y] = (desired_rot_vel[Y] - angular_velocity_vector[Y])
                     * INERTIA/DELTA_T;
    moment_body[Z] = (desired_rot_vel[Z] - angular_velocity_vector[Z])
                     * INERTIA/DELTA_T;
```

APPENDIX B - rwa_aerodyn.c

```
    moment_body[X] = min (MAX_TORQUE, moment_body[X]);
    moment_body[Y] = min (MAX_TORQUE, moment_body[Y]);
    moment_body[Z] = min (MAX_TORQUE, moment_body[Z]);

    moment_body[X] = max (-MAX_TORQUE, moment_body[X]);
    moment_body[Y] = max (-MAX_TORQUE, moment_body[Y]);
    moment_body[Z] = max (-MAX_TORQUE, moment_body[Z]);

    interact_with_ground();
    vec_add (force_body, ground_force,force_body);
    vec_add (force_body, gravity_force_body,force_body);
    vec_add (force_body, force_ground_effect, force_body);

    send_to_dynamics_kinematics ();
    vehicle_update ();
}


/**********************************************************************
 * for tweaking purposes, use parameter file for constants            *
 **********************************************************************/
aerodyn_read_simple_constants (fn)
char *fn;
{
    char *strtok ();
    FILE *fp;
    char s[80];

    if ((fp = FOPEN (fn, "r")) == NULL)
    {
      printf ("no tweakable constants file; using defaults\n", fn);
      return (-1);
    }
    else
      printf ("Reading tweakable constants file: %s\n", fn);

    while (FGETS (s, 80, fp) != NULL)
    {
       char *str;
       switch (s[0])  /* check for comments or blank lines */
         {
         case '#':
         case ' ':
         case '\n':
         case '\t':
           continue;
       }

       str = strtok (s, " \t");

       if (strcmp (str, "H_K1") == 0)
       {
           sscanf (strtok (0, " \t"), "%lf", &H_K1);
           continue;
```

```
}

if (strcmp (str, "H_K2") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_K2);
    continue;
}

if (strcmp (str, "H_K7") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_K7);
    continue;
}

if (strcmp (str, "H_K8") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_K8);
    continue;
}

if (strcmp (str, "H_KP") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_KP);
    continue;
}

if (strcmp (str, "H_KPR") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_KPR);
    continue;
}
if (strcmp (str, "H_KY") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_KY);
    continue;
}

if (strcmp (str, "H_KH") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_KH);
    continue;
}

if (strcmp (str, "H_FWD_MUL") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_FWD_MUL);
    continue;
}

if (strcmp (str, "H_COLL_MUL") == 0)
{
    sscanf (strtok (0, " \t"), "%lf", &H_COLL_MUL);
    continue;
}
```

```
    if (strcmp (str, "H_CHH") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &H_CHH);
        continue;
    }

    if (strcmp (str, "H_CL") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &H_CL);
        continue;
    }

    if (strcmp (str, "MAX_FORCE") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &MAX_FORCE);
        continue;
    }

    if (strcmp (str, "MAX_TORQUE") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &MAX_TORQUE);
        continue;
    }

    if (strcmp (str, "MASS") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &MASS);
        continue;
    }

    if (strcmp (str, "INERTIA") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &INERTIA);
        continue;
    }

    if (strcmp (str, "H_SIDE_MUL") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &H_SIDE_MUL);
        continue;
    }

    if (strcmp (str, "DEAD_ZONE") == 0)
    {
        sscanf (strtok (0, " \t"), "%lf", &DEAD_ZONE);
        continue;
    }

    /* if got here -- mistake */
    printf ("ERROR: Unknown constant %s in %s\n", str, fn);
  }
  FCLOSE (fp);
  printf ("done reading constants file\n");
/*   aerodyn_dump_simple_constants ();*/
  return (1);
```

```
}

aerodyn_dump_control_inputs ()
{
    printf ("collective = %.2lf\tcyclic_roll = %.2lf\tcyclic_pitch = %.2lf\n",
         collective, cyclic_roll, cyclic_pitch);
    printf ("pedal = %.2lf\n", pedal);
    aerodyn_debug = aerodyn_debug ? 0 : 1;
    printf ("aerodyn_debug is %s\n", aerodyn_debug ? "on" : "off");
}

aerodyn_dump_simple_constants ()
{
    printf ("Aerodyn simple constants:\n");
    printf ("\tH_K1:\t%.2lf\n", H_K1);
    printf ("\tH_K2:\t%.2lf\n", H_K2);
    printf ("\tH_K7:\t%.2lf\n", H_K7);
    printf ("\tH_K8:\t%.2lf\n", H_K8);
    printf ("\tH_KP:\t%.2lf\n", H_KP);
    printf ("\tH_KPR:\t%.2lf\n", H_KPR);
    printf ("\tH_KY:\t%.2lf\n", H_KY);
    printf ("\tH_KH:\t%.2lf\n", H_KH);
    printf ("\tH_FWD_MUL:\t%.2lf\n", H_FWD_MUL);
    printf ("\tH_SIDE_MUL:\t%.2lf\n", H_SIDE_MUL);
    printf ("\tH_COLL_MUL:\t%.2lf\n", H_COLL_MUL);
    printf ("\tH_CHH:\t%.2lf\n", H_CHH);
    printf ("\tH_CL:\t%.2lf\n", H_CL);
    printf ("\tMAX_FORCE:\t%.2lf\n", MAX_FORCE);
    printf ("\tMAX_TORQUE:\t%.2lf\n", MAX_TORQUE);
    printf ("\tMASS:\t%.2lf\n", MASS);
    printf ("\tINERTIA:\t%.2lf\n", INERTIA);
    printf ("\tDEAD_ZONE:\t%.2lf\n", DEAD_ZONE);
}

set_selected_model (model)
int model;
{
    switch (model)
    {
      case COMPLEX_MODEL:
      printf ("switching to complex model, logarithmic collective\n");
      funny_little_kludge = 1;/* logarithmic collective */
      selected_model = model;
      break;
      case SIMPLE_MODEL:
      printf ("switching to simple model, linear collective\n");
      funny_little_kludge = 0;/* linear collective */
      selected_model = model;
      break;
      case STEALTH_MODEL:
      printf ("switching to stealth model, linear collective\n");
      funny_little_kludge = 0;/* linear collective */
      selected_model = model;
      break;
      default:
```

```
      printf ("invalid selected model %d\n", model);
      printf ("using default complex model\n");
      selected_model = COMPLEX_MODEL;
      break;
    }
}


get_selected_model ()
{
    return (selected_model);
}

indicate_selected_model (model)
int model;
{
    switch (model)
    {
      case COMPLEX_MODEL:
      printf ("using complex model\n");
      break;
      case SIMPLE_MODEL:
      printf ("using simple model\n");
      break;
      case STEALTH_MODEL:
      printf ("using stealth model\n");
        allow_takeoff = TRUE;
      break;
      default:
      printf ("invalid selected model %d\n", model);
      printf ("using default complex model\n");
      break;
    }
}

set_takeoff_status (status)
int status;
{
    allow_takeoff = status;
}
```

Appendix C - Source code listing for rwa_engine.c.

The following appendix contains the source code listing for rwa_engine.c for convenience in document maintenance and understanding of the CSU.

APPENDIX C - rwa_engine.c

```
/* $Header: /a3/adst-cm/RWA/simnet/vehicle/rwa/src/RCS/rwa_engine.c,v 1.5 1992/1
2/21 22:15:59 cm-adst Exp $ */
/*
 * $Log: rwa_engine.c,v $
 * Revision 1.5  1992/12/21  22:15:59  cm-adst
 * R. Branson's flight changes.  These changes will become
 * BDS-D 1.1.1.  This change was turned over by C. Swanson.
 *
 * Revision 1.1  1992/10/07  19:00:23  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/simnet/vehicle/rwa/src/RCS/rwa_
engine.c,v 1.5 1992/12/21 22:15:59 cm-adst Exp $";

/*************************************************************************
 *
 * Revisions:
 *
 *      Version     Date       Author          Title                   SP/CR Number
 *      _______     ________   __________      ___________________     ____________
 *
 *      1.2         10/09/92   R. Branson      Data File Initiali-
 *                                               zation
 *      1.3         10/16/92   R. Branson      Data filenames changed
 *                                               to eight characters
 *      1.4         10/30/92   R. Branson      Added pathname to data
 *                                               directory
 *      1.5         01/19/93   P.Desmeules     Increased the size of the      31
 *                                             fgets to make sure the
 *                                             whole line is read in.
 *
 *
 *************************************************************************/

/*************************************************************************
 *
 *        SP/CR No.        Description of Modification
 *        _________        ______________________________________________
 *
 *
 *                         Hard coded defines changed to array elements.
 *                         Engine data array added.
 *                         Engine initialization data array added.
 *                         Engine status data array added.
 *                         Added file for engine data, engine initialization
 *                            data, and engine status data to the "engine_init"
 *                            function
 *
 *                         Added "/simnet/data/" to each data file pathname.
 *
 *************************************************************************/

/**************************************************************
 *                                                            *
 * FILE:          rwa_engine.c                                *
 * AUTHOR:        James Chung                                 *
```

APPENDIX C - rwa_engine.c

```
 * MAINTAINER:     James Chung                                         *
 * HISTORY:        4/19/89  james: Creation                            *
 *                                                                     *
 *                                                                     *
 * Copyright (c) 1989 BBN Systems and Technologies Corporation         *
 * All rights reserved.                                                *
 *                                                                     *
 * Interim engine model for the generic rotary-wing aircraft           *
 * with power characteristics similar to the General                   *
 * T700-GE-701 turboshaft engine. The T700 is rated at a               *
 * maximum continuous power of 1510 shp at sea-level.                  *
 * Two (2) T700s power the AH-64 Apache attack helicopter.             *
 ***********************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_dfns.h"
#include "sim_macros.h"
#include "sim_types.h"
#include "libsound.h"
#include "rwa_soun_dfn.h"
#include "rwa_meter.h"
#include "rwa_cntrl.h"
#include "libmun.h"
#include "failure.h"
#include "libfail.h"

/*
 * Debug macro
 */
#ifdef FILEDBG
#define P(a)        a
#else
#define P(a)
#endif

/* Once the engine or transmission has been damaged, there is a chance that
   the engine/transmission will seize due to too many particle fragments
   accumulating in the respective oil system.  These are "secondary" events.
                                   12-10-90    pjm   */

#define DO_CFAIL  TRUE  /* do combat damage simulation */
#define DO_SFAIL  TRUE  /* do stochastic failure simulation */

static REAL engine_data[20] = {
      1030.55,       0.05,       0.05,     1031.6,       25.0,
         1.2,     1200.0,       0.16438,    2.130,       34.0,
         7.0,      100.0,     153.8461539,  0.0,          0.0,
           0.0,        0.0,        0.0,        0.0,         0.0
         } ;

static REAL engine_init_data[10] = {
         0.0,        0.0,        0.0,        0.0,        0.0,
         1.0,        0.0,        0.0,        0.0,        0.0
```

```
        } ;

static int engine_stat_data[10] = {
        0,          0,          1,          1,          2,
        0,          0,          0,          0,          0
        } ;

#define GOVERNOR_ENGINE_SPEED_SETTING     engine_data[ 0]
#define GOVERNOR_P_GAIN                   engine_data[ 1]
#define GOVERNOR_I_GAIN                   engine_data[ 2]
#define MAX_ENGINE_TORQUE                 engine_data[ 3]
#define MIN_ENGINE_LOAD_TORQUE            engine_data[ 4]
#define MAX_ENGINE_PERCENT_POWER          engine_data[ 5]
#define ENGINE_TORQUE_INTERCEPT           engine_data[ 6]
#define ENGINE_TORQUE_SLOPE               engine_data[ 7]
#define NOSE_GEARBOX_RATIO                engine_data[ 8]
#define MAIN_ROTOR_GEAR_RATIO             engine_data[ 9]
#define TAIL_ROTOR_GEAR_RATIO             engine_data[10]
#define POWERTRAIN_INERTIA                engine_data[11]
#define MAX_FUELFLOW                      engine_data[12]

/* (seconds/tick) / (seconds/hour) = (hours/tick) */
#define HOURS_PER_TICK   ( DELTA_T / 3600.0 )
static REAL hours_of_flight;
static int minutes_of_flight, old_minutes_of_flight;
static BOOLEAN engine_is_damaged, transmission_is_damaged;

/****** engine noise stuff *****/
#define ORIGINAL  0
#define BOTH_DISABLED    1
#define CHANGE_ROTOR     2
#define CHANGE_ENGINE    3
#define CHANGE_BOTH      4
static int engine_sound_type = CHANGE_BOTH;
static int engine_oscillation[2], rotor_oscillation[2];

#define MIN_ROTOR_SOUND        105
#define MAX_ROTOR_SOUND        120
#define ROTOR_SOUND_RANGE      (MAX_ROTOR_SOUND - MIN_ROTOR_SOUND)
#define MIN_TURBINE_SOUND      95
#define MAX_TURBINE_SOUND      126
#define TURBINE_SOUND_RANGE    (MAX_TURBINE_SOUND - MIN_TURBINE_SOUND)

static REAL turbine_speed;
static REAL engine_speed;      /* Nose gearbox output shaft */
static REAL engine_load_torque;
static REAL engine_percent_torque;
static REAL engine_drive_torque;
static REAL main_rotor_shaft_speed;
static REAL main_rotor_drive_torque;
static REAL tail_rotor_shaft_speed;
static REAL tail_rotor_drive_torque;
static REAL powertrain_percent_shaft_speed;
static REAL last_percent_shaft_speed;
static REAL last_percent_torque;
```

APPENDIX C - rwa_engine.c

```
static REAL fuel_flow;
static REAL engine_power;
static REAL integrator_gain;
static REAL gov_p_gain;
static REAL gov_i_gain;

static int number_of_engines; /* Working */
static int engine_status;

/* Flag used to determine if the engine is starting.  Sounds for the engine
   and rotors are more "realistic."  Starting engine speed is 0 instead of
   GOVERNOR_ENGINE_SPEED_SETTING, and since engine_power then maxes out
   (causes "torque" to flash) a check is done and temporarily forces the
   torque percentage to be equal to 1.
                                    11-8-89           Paul J. Metzger */
static int starting_engine;

void    engine_simul (main_rotor_load, tail_rotor_load, altitude)
REAL    main_rotor_load, tail_rotor_load, altitude;
{
    REAL    tail_rotor_engine_load;
    REAL    main_rotor_engine_load;
    REAL    temp_percent;
    int     temp_sound;

    main_rotor_engine_load = main_rotor_load / MAIN_ROTOR_GEAR_RATIO;
    tail_rotor_engine_load = tail_rotor_load / TAIL_ROTOR_GEAR_RATIO;

    engine_load_torque = main_rotor_engine_load + tail_rotor_engine_load;
    if (engine_load_torque < MIN_ENGINE_LOAD_TORQUE)
      engine_load_torque = MIN_ENGINE_LOAD_TORQUE;

    engine_power = gov_p_gain *
      (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);

    if (engine_status == WORKING)
    {
      integrator_gain += gov_i_gain *
          (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
      if (integrator_gain > 0.5)
          integrator_gain = 0.5;
      else if (integrator_gain < -0.5)
          integrator_gain = -0.5;

      engine_power += integrator_gain;
    }
    else                /* Damaged */
    {
      integrator_gain = 0.0;
      if (engine_power > 0.7)
          engine_power = 0.7;
    }

    if (engine_power > MAX_ENGINE_PERCENT_POWER)
      engine_power = MAX_ENGINE_PERCENT_POWER;
```

```
if (engine_power < 0.0)
  engine_power = 0.0;

if (fuel_level_empty ())  /* Out of gas */
{
  engine_power = 0.0;
  engine_speed = 0.0;
}


engine_drive_torque = engine_power * number_of_engines *
  (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE * engine_speed);

engine_percent_torque = engine_drive_torque /
  (MAX_ENGINE_TORQUE * number_of_engines);

if (engine_status == WORKING)
  engine_speed += (engine_drive_torque - engine_load_torque)
      / POWERTRAIN_INERTIA;

if (engine_speed < 0.0)
  engine_speed = 0.0;

turbine_speed = engine_speed * NOSE_GEARBOX_RATIO;
main_rotor_shaft_speed = engine_speed / MAIN_ROTOR_GEAR_RATIO;
tail_rotor_shaft_speed = engine_speed / TAIL_ROTOR_GEAR_RATIO;
powertrain_percent_shaft_speed = engine_speed /
  GOVERNOR_ENGINE_SPEED_SETTING;
tail_rotor_drive_torque = tail_rotor_load;  /* Always have tail rotor */
main_rotor_drive_torque = (engine_drive_torque - tail_rotor_engine_load)
  * MAIN_ROTOR_GEAR_RATIO;
if (main_rotor_drive_torque < 0.0)
  main_rotor_drive_torque = 0.0;

fuel_flow = engine_percent_torque * MAX_FUELFLOW;

if (engine_status == BROKEN)/* crippled condition */
{
  sound_stop_cont_sound (SOUND_OF_STOP_ENGINE, SOUND_OF_VARY_ENGINE);
  sound_stop_cont_sound (SOUND_OF_STOP_ROTOR, SOUND_OF_VARY_ROTOR);
  fuel_flow *= 50.0;      /* fuel leak */
}

if (starting_engine)
{
  if (engine_percent_torque - .01 < .0001)  /* within a delta */
      starting_engine = FALSE;
  else
      engine_percent_torque = .01;
}

fuel_used_by_engine (fuel_flow / 3600.0 * DELTA_T);

meter_torque_set (engine_percent_torque);
```

APPENDIX C - rwa_engine.c

```
    meter_rpm_set (powertrain_percent_shaft_speed);

    hours_of_flight += HOURS_PER_TICK;
    minutes_of_flight = (int) (hours_of_flight * 60);
#if DO_SFAIL
    if (minutes_of_flight > old_minutes_of_flight)
    {
      sfail_event_occurred (SFAIL_EVENT_MILEAGE);
      if (engine_is_damaged)
          sfail_event_occurred (SFAIL_SECONDARY_EVENT_ENGINE);
      if (transmission_is_damaged)
          sfail_event_occurred (SFAIL_SECONDARY_EVENT_TRANSMISSION);
      old_minutes_of_flight = minutes_of_flight;
    }
#endif

    if (!fuel_level_empty ())
    {
      switch (engine_sound_type)
      {
      case CHANGE_ENGINE:
          if (abs (powertrain_percent_shaft_speed
                 - last_percent_shaft_speed) > 0.025)
          {
            /* rotor sounds depend on RPMs
             * (powertrain_percent_shaft_speed) */
            temp_percent = max (0.01, powertrain_percent_shaft_speed);
            sound_make_cont_sound (SOUND_OF_START_ROTOR, SOUND_OF_VARY_ROTOR,
                                   SOUND_OF_STOP_ROTOR, temp_percent);
            last_percent_shaft_speed = powertrain_percent_shaft_speed;
          }

          if (abs (engine_percent_torque - last_percent_torque) > 0.025)
          {
            /* engine sounds depend on torque (engine_percent_torque) */
            temp_percent = max (0.01, engine_percent_torque);
            sound_make_cont_sound (SOUND_OF_START_ENGINE, SOUND_OF_VARY_ENGINE,
                                   SOUND_OF_STOP_ENGINE, temp_percent);
            last_percent_torque = engine_percent_torque;
          }
          break;

      case ORIGINAL:
          if (abs (powertrain_percent_shaft_speed
                 - last_percent_shaft_speed) > 0.025)
          {
            /* rotor sounds depend on RPMS
             * (powertrain_percent_shaft_speed) */
            temp_percent = max (0.01, powertrain_percent_shaft_speed);
            sound_make_cont_sound (SOUND_OF_START_ROTOR, SOUND_OF_VARY_ROTOR,
                                   SOUND_OF_STOP_ROTOR, temp_percent);
            sound_make_cont_sound (SOUND_OF_START_ENGINE, SOUND_OF_VARY_ENGINE,
                                   SOUND_OF_STOP_ENGINE, temp_percent);
            last_percent_shaft_speed = powertrain_percent_shaft_speed;
          }
```

```
        break;

    case CHANGE_BOTH:
        /* Try the following, as per Perc's directions:  vary both the
         * rotor and engine with torque, but have the rotor range be from
         * 105 to 120, and the turbine range from 95 to 126.
         *
         * The rotor sound range is 15 points (120-105), so the % torque is
         * multiplied by 15, then added to an offset of 105.
         *
         * The turbine sound range is 31 points (126-95), so the % torque is
         * multiplied by 31, then added to an offset of 105.
         *
         * 11-17-90    PJM           */
        if (abs (engine_percent_torque - last_percent_torque) > 0.025)
        {
          /* both sounds depend on torque */
          temp_sound = (int) (engine_percent_torque * ROTOR_SOUND_RANGE) +
              MIN_ROTOR_SOUND;
          if (temp_sound > MAX_ROTOR_SOUND)
              temp_sound = MAX_ROTOR_SOUND;

          /* We check to see if the sounds are oscillating.  This */
          /* event occurs while at the extreme torque edges of */
          /* the hover hold mode, when we're trying to break */
          /* hold.                    2-15-91   PJM       */

          if (temp_sound != rotor_oscillation[1])
               sound_make_arg_sound (SOUND_OF_VARY_ROTOR, temp_sound);

          rotor_oscillation[1] = rotor_oscillation[0];
          rotor_oscillation[0] = temp_sound;

          temp_sound = (int) (engine_percent_torque *
              TURBINE_SOUND_RANGE) + MIN_TURBINE_SOUND;
          if (temp_sound > MAX_TURBINE_SOUND)
              temp_sound = MAX_TURBINE_SOUND;

          if (temp_sound != engine_oscillation[1])
               sound_make_arg_sound (SOUND_OF_VARY_ENGINE, temp_sound);

          engine_oscillation[1] = engine_oscillation[0];
          engine_oscillation[0] = temp_sound;

          last_percent_torque = engine_percent_torque;
        }
        break;

    case CHANGE_ROTOR:
        if (abs (engine_percent_torque - last_percent_torque) > 0.025)
        {
          /* rotor sounds depend on torque */
          temp_sound = (int) (engine_percent_torque * ROTOR_SOUND_RANGE) +
              MIN_ROTOR_SOUND;
          if (temp_sound > MAX_ROTOR_SOUND)
```

```
                temp_sound = MAX_ROTOR_SOUND;
            sound_make_arg_sound (SOUND_OF_VARY_ROTOR, temp_sound);
            sound_stop_cont_sound (SOUND_OF_STOP_ENGINE,
                                   SOUND_OF_VARY_ENGINE);
            last_percent_torque = engine_percent_torque;
          }
          break;

      case BOTH_DISABLED:
          sound_stop_cont_sound (SOUND_OF_STOP_ENGINE, SOUND_OF_VARY_ENGINE);
          sound_stop_cont_sound (SOUND_OF_STOP_ROTOR, SOUND_OF_VARY_ROTOR);
          break;
      }
    }

}

REAL    engine_get_rotor_percent_shaft_speed ()
{
    return (powertrain_percent_shaft_speed);
}

void    engine_damage_engine_oil ()
{
#if DO_CFAIL
    controls_start_failure_lamp_flashing (MASTER_CAUTION);
    controls_start_failure_lamp_flashing (ENGINE_FAILURE);
#endif
    engine_is_damaged = TRUE;
}

void    engine_repair_engine_oil ()
{
#if DO_CFAIL
    controls_failure_lamp_off (ENGINE_FAILURE);
    engine_is_damaged = FALSE;
#endif
}

void    engine_break_engine ()
{
    engine_status = BROKEN;
    engine_speed = 0.0;
    number_of_engines = 1;
}

void    engine_repair_engine ()
{
    engine_repair_engine_oil ();
    engine_status = WORKING;
    number_of_engines = 2;
}

void    engine_damage_transmission_filter ()
{
```

APPENDIX C - rwa_engine.c

```
#if DO_SFAIL
    controls_start_failure_lamp_flashing (MASTER_CAUTION);
    controls_start_failure_lamp_flashing (TRANSMISSION_FAILURE);
    transmission_is_damaged = TRUE;
#endif
}

void    engine_repair_transmission_filter ()
{
#if DO_SFAIL
    controls_failure_lamp_off (TRANSMISSION_FAILURE);
    transmission_is_damaged = FALSE;
#endif
}

void    engine_break_transmission ()
{
#if DO_SFAIL
    engine_break_engine ();    /* engine has seized */
#endif
}

void    engine_repair_transmission ()
{
#if DO_SFAIL
    engine_repair_transmission_filter ();
    engine_repair_engine ();
#endif
}


void    engine_init ()
{
      int    i;
      int    data_init;
      float  data_tmp;
      char   descript[80];
      FILE   *fp;

      P(printf("$$$$ RWA ENGINE file data $$$$\n"););

/*  DEFAULT DATA FOR rwa_engine.c READ FROM FILE                    */
      fp = fopen("/simnet/data/rwa_engn.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rwa_engn.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            engine_data[i] = data_tmp;
```

APPENDIX C - rwa_engine.c

```
            fgets(descript, 80, fp);
            P(printf("engine_data(%3d) is%11.3f %s", i, engine_data[i],
                        descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT DATA FOR rwa_engine.c READ FROM FILE                    */

/*  DEFAULT INITIALIZATION DATA FOR rwa_engine.c READ FROM FILE         */
    fp = fopen("/simnet/data/rw_en_in.d","r");
    if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rw_en_in.d\n");
            exit();
    }

    rewind(fp);

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
            engine_init_data[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("engine_init_data(%3d) is%11.3f %s", i,
                    engine_init_data[i], descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT INITIALIZATION DATA FOR rwa_engine.c READ FROM FILE     */

/*  DEFAULT STATUS DATA FOR rwa_engine.c READ FROM FILE                 */
    fp = fopen("/simnet/data/rw_en_st.d","r");
    if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rw_en_st.d\n");
            exit();
    }

    rewind(fp);

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%d", &data_init) != EOF){
            engine_stat_data[i] = data_init;
            fgets(descript, 80, fp);
            P(printf("engine_stat_data(%3d) is%11d %s", i,
                    engine_stat_data[i], descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT STATUS DATA FOR rwa_engine.c READ FROM FILE             */
```

```
    gov_p_gain =                 GOVERNOR_P_GAIN;
    gov_i_gain =                 GOVERNOR_I_GAIN;
    engine_power =               engine_init_data[ 0];
    engine_percent_torque =      engine_init_data[ 1];
    engine_speed =               engine_init_data[ 2];
    integrator_gain =            engine_init_data[ 3];
    last_percent_shaft_speed =   engine_init_data[ 4];
    last_percent_torque =        engine_init_data[ 5];
    hours_of_flight =            engine_init_data[ 6];
    minutes_of_flight =          engine_stat_data[ 0];
    old_minutes_of_flight =      engine_stat_data[ 1];
    engine_status =              engine_stat_data[ 2];
    starting_engine =            engine_stat_data[ 3];
    number_of_engines =          engine_stat_data[ 4];
    engine_is_damaged =          engine_stat_data[ 5];
    transmission_is_damaged =    engine_stat_data[ 6];

#if DO_CFAIL
    fail_init_failure (motiveOilLeak, engine_damage_engine_oil,
                engine_repair_engine_oil, NO_SELF_REPAIR, noncritKill);
    fail_init_failure (motiveEngineMajor, engine_break_engine,
                   engine_repair_engine, NO_SELF_REPAIR, mobilityKill);
#endif

#if DO_SFAIL
    fail_init_failure (motiveTransFluidFilter,
       engine_damage_transmission_filter, engine_repair_transmission_filter,
                   NO_SELF_REPAIR, noncritKill);
    fail_init_failure (motiveTransmissionMajor, engine_break_transmission,
             engine_repair_transmission, NO_SELF_REPAIR, mobilityKill);
#endif
}

void    engine_debug_print ()
{
    printf ("rpm = %f\n  rps = %f\n  ps = %f\n etq = %f\n mrt = %f\n",
          powertrain_percent_shaft_speed, engine_speed,
          engine_power, engine_drive_torque, main_rotor_drive_torque);
}

REAL    engine_get_speed ()
{
    return (engine_speed);
}

void    engine_toggle_sound ()
{
    if ((engine_sound_type - 1) < ORIGINAL)
      engine_sound_type = CHANGE_BOTH;
    else
      engine_sound_type--;

    switch (engine_sound_type)
    {
```

APPENDIX C - rwa_engine.c

```
    case ORIGINAL:
      printf ("Rotor: RPM    Engine: RPM\n");
      break;
    case CHANGE_ROTOR:
      printf ("Rotor: TORQUE    Engine: DISABLED\n");
      break;
    case CHANGE_ENGINE:
      printf ("Rotor: RPM    Engine: TORQUE\n");
      break;
    case CHANGE_BOTH:
      printf ("Rotor: TORQUE    Engine: TORQUE\n");
      break;
    case BOTH_DISABLED:
      printf ("Rotor: DISABLED    Engine: DISABLED\n");
      break;
    }
}

REAL    engine_get_hours_of_flight ()
{
    return (hours_of_flight);
}

int     engine_get_minutes_of_flight ()
{
    return (minutes_of_flight);
}
```

Appendix D- Source code listing for rwa_kinemat.c.

The following appendix contains the source code listing for rwa_kinemat.c for convenience in document maintenance and understanding of the CSU.

APPENDIX D - rwa_kinemat.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_kinemat.c,v
1.6 1993/01/28 23:33:00 cm-adst Exp $ */
/*
 * $Log: rwa_kinemat.c,v $
 * Revision 1.6  1993/01/28  23:33:00  cm-adst
 * P. DesMeueles's changes for spcr 31
 *
 * Revision 1.5  1992/12/21  22:16:49  cm-adst
 * R. Branson's flight changes.  These changes will become
 * BDS-D 1.1.1.  This change was turned over by C. Swanson.
 *
 * Revision 1.1  1992/10/07  19:00:23  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_kinemat.c,v 1.6 1993/01/28 23:33:00
cm-adst Exp $";

/*************************************************************************
 *
 * Revisions:
 *
 *     Version        Date        Author      Title                    SP/CR Number
 *     _______      ________    ________    ___________________      ____________
 *
 *     1.2          10/09/92    R. Branson  Data File Initiali-
 *                                          zation
 *     1.3          10/16/92    R. Branson  Data filenames changed
 *                                            to eight characters
 *     1.4          10/30/92    R. Branson  Added pathname to data
 *                                            directory
 *     1.5          01/19/93    P.Desmeules Increased the size of the     31
 *                                          fgets to make sure the
 *                                          whole line is read in.
 *                                                                        85
 *     1.5          03/04/93    P.Desmeules Fix value of
 *                                          DISPLAY_SPEED_LIMIT
 *
 *************************************************************************/

/*************************************************************************
 *
 *       SP/CR No.         Description of Modification
 *     _____________     ___________________________________________
 *
 *
 *                       Hard coded defines changed to array element.
 *                       Kinemat data array added.
 *                       Kinemat initialization array added.
 *                       Added file read for kinemat data and kinemat initiali-
 *                          zation data to the "veh_spec_kinematics_init"
 *                          function.
 *
 *                       Added "/simnet/data/" to each data file pathname.
 *
 *************************************************************************/
```

APPENDIX D - rwa_kinemat.c

```
/*****************************************************************
 *                                                               *
 * FILE:          rwa_kinemat.c                                  *
 * AUTHOR:        Bryant Collard                                 *
 * MAINTAINER:    Bryant Collard                                 *
 * PURPOSE:       This file contains routines which process      *
 *                information generated in the dynamics and      *
 *                kinematics software to generate data needed    *
 *                specifically for the rotary wing aircraft.     *
 * HISTORY:       03/03/89 bryant: Creation                      *
 *                05/15/89 james:  Modified for RWA              *
 *                                                               *
 *                                                               *
 * Copyright (c) 1989 BBN Systems and Technologies, Inc.         *
 * All rights reserved.                                          *
 *                                                               *
 *****************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "sim_macros.h"

#include "libmatrix.h"
#include "librotate.h"
#include "vehicle.h"
#include "std_atm.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

#define GRAV_CONSTANT          kinemat_data[ 0]

#define SIN_AOA_LIMIT          kinemat_data[ 1]
#define COS_AOA_LIMIT          kinemat_data[ 2]
#define SIN_YAW_LIMIT          kinemat_data[ 3]
#define COS_YAW_LIMIT          kinemat_data[ 4]

#define DISPLAY_SPEED_LIMIT    kinemat_data[ 5]

static VECTOR pos_unit_vel;
static VECTOR neg_unit_vel;
static REAL sin_aoa;
static REAL cos_aoa;
static REAL sin_yaw;
static REAL cos_yaw;
```

```
static REAL altitude;
static REAL body_pitch;
static REAL body_pitch_offset;
static REAL velocity_pitch;
static REAL roll;
static REAL heading;
static REAL true_airspeed;
static REAL indicated_airspeed;
static REAL g_force;
static REAL vertical_speed;
static REAL *ang_vel;
static REAL *velocity_vector;
static VECTOR gravity;
static VECTOR norm_vel;
static T_MATRIX velocity_to_body;

/*
* SPCR 85 - Fix the value of DISPLAY_SPEED_LIMIT (element 5) from 0.0 to 5.0
*/
static REAL kinemat_data[20] = {
      9.81,  0.642787610,  0.766044443,  0.642787610,  0.766044443,
      5.0,        0.0,         0.0,          0.0,          0.0,
      0.0,        0.0,         0.0,          0.0,          0.0,
      0.0,        0.0,         0.0,          3.0,          0.0
      } ;

static REAL kinemat_init_data[30] = {
      0.0,        1.0,         0.0,          0.0,         -1.0,
      0.0,        0.0,         1.0,          0.0,          1.0,
      0.0,        0.0,         0.0,          0.0,          0.0,
      0.0,        0.0,         0.0,          1.0,          0.0,
      0.0,        0.0,        -1.0,          0.0,          1.0,
      0.0,        0.0,         0.0,          0.0,          0.0
      } ;

/*****************************************************************
 *                                                     *
 * ROUTINE: veh_spec_kinematics_init                         *
 * PARAMETERS:    none                                       *
 * RETURNS: none                                       *
 * PURPOSE: This routine initializes vehicle specific *
 *          kinematics parameters.                           *
 *                                                     *
 *****************************************************************/

void veh_spec_kinematics_init ()
{
/*  DEFAULT DATA FOR rwa_kinemat.c READ FROM FILE     */
      int   i;
      float   data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ RWA KINEMATICS file data $$$$\n"););
```

APPENDIX D - rwa_kinemat.c

```
    fp = fopen("/simnet/data/rwa_kine.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/rwa_kine.d\n");
          exit();
    }

    rewind(fp);

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          kinemat_data[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kinemat_data(%3d) is%11.3f %s", i, kinemat_data[i],
                      descript););
          ++i;
    }

    fclose(fp);

/*  END DEFAULT DATA FOR rwa_kinemat.c READ FROM FILE      */

/*  DEFAULT INITIALIZATION DATA FOR rwa_kinemat.c READ FROM FILE      */

    fp = fopen("/simnet/data/rw_ki_in.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/rw_ki_in.d\n");
          exit();
    }

    rewind(fp);

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          kinemat_init_data[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kinemat_init_data(%3d) is%11.3f %s", i,
                      kinemat_init_data[i], descript););
          ++i;
    }

    fclose(fp);

/*  END DEFAULT INITIALIZATION DATA FOR rwa_kinemat.c READ FROM FILE      */

    pos_unit_vel[Y] =    kinemat_init_data[ 1];
    pos_unit_vel[Z] =    kinemat_init_data[ 2];
    neg_unit_vel[X] =    kinemat_init_data[ 3];
    neg_unit_vel[Y] =    kinemat_init_data[ 4];
    neg_unit_vel[Z] =    kinemat_init_data[ 5];
    sin_aoa =            kinemat_init_data[ 6];
```

```
    cos_aoa =              kinemat_init_data[ 7];
    sin_yaw =              kinemat_init_data[ 8];
    cos_yaw =              kinemat_init_data[ 9];
    altitude =             kinemat_init_data[10];
    body_pitch =           kinemat_init_data[11];
    body_pitch_offset =    kinemat_init_data[12];
    velocity_pitch =       kinemat_init_data[13];
    roll =                 kinemat_init_data[14];
    heading =              kinemat_init_data[15];
    true_airspeed =        kinemat_init_data[16];
    indicated_airspeed =   kinemat_init_data[17];
    g_force =              kinemat_init_data[18];
    vertical_speed =       kinemat_init_data[19];
    ang_vel = vehicle_angular_velocity ();
    velocity_vector = vehicle_velocity();
    gravity[X] =           kinemat_init_data[20];
    gravity[Y] =           kinemat_init_data[21];
    gravity[Z] =           kinemat_init_data[22];
    norm_vel[X] =          kinemat_init_data[23];
    norm_vel[Y] =          kinemat_init_data[24];
    norm_vel[Z] =          kinemat_init_data[25];
    mat_ident (velocity_to_body);
}

/*****************************************************************
 *                                                   *
 * ROUTINE: veh_spec_kinematics_simul                      *
 * PARAMETERS:    none                                     *
 * RETURNS: none                                     *
 * PURPOSE: This routine finds  vehicle specific kinematics *
 *          parameters.                              *
 *                                                   *
 *****************************************************************/

void veh_spec_kinematics_simul ()
{
    REAL *velocity;
    REAL temp, temp2;
    REAL *position;
    T_MAT_PTR body_to_world;

    position = rotate_get_loc (world (), hull ());
    altitude = position[Z];
    if (altitude < 0.0)
        altitude = 0.0;
/*    velocity = vehicle_velocity (); */
    velocity = velocity_vector;
    true_airspeed = sqrt (velocity[X] * velocity[X] + velocity[Y] * velocity[Y]
            + velocity[Z] * velocity[Z]);
    indicated_airspeed = true_airspeed * sqrt (air_density (altitude) /
            air_density(0.0));
    if (true_airspeed < E_MILLI)
    {
        norm_vel[X] = 0.0;
        norm_vel[Y] = 1.0;
```

APPENDIX D - rwa_kinemat.c

```
        norm_vel[Z] = 0.0;
    }
    else
    {
        norm_vel[X] = velocity[X] / true_airspeed;
        norm_vel[Y] = velocity[Y] / true_airspeed;
        norm_vel[Z] = velocity[Z] / true_airspeed;
    }
    if (norm_vel[Z] - 1.0 > -E_NANO)
    {
        sin_aoa = -1.0;
        cos_aoa = 0.0;
        sin_yaw = 0.0;
        cos_yaw = 1.0;
    }
    else if (norm_vel[Z] + 1.0 < E_NANO)
    {
        sin_aoa = 1.0;
        cos_aoa = 0.0;
        sin_yaw = 0.0;
        cos_yaw = 1.0;
    }
    else
    {
        sin_aoa = -norm_vel[Z];
        cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] * norm_vel[Y]);
        sin_yaw = norm_vel[X] / cos_aoa;
        cos_yaw = norm_vel[Y] / cos_aoa;
    }
/*
    if (sin_aoa > SIN_AOA_LIMIT)
    {
        temp = COS_AOA_LIMIT;
        velocity_to_body[1][2] = -SIN_AOA_LIMIT;
    }
    else if (sin_aoa < -SIN_AOA_LIMIT)
    {
        temp = COS_AOA_LIMIT;
        velocity_to_body[1][2] = SIN_AOA_LIMIT;
    }
    else
    {
*/
        temp = cos_aoa;
        velocity_to_body[1][2] = -sin_aoa;
/*
    }
    if (cos_yaw < COS_YAW_LIMIT)
    {
        velocity_to_body[0][0] = COS_YAW_LIMIT;
        if (sin_yaw > 0)
            velocity_to_body[0][1] = -SIN_YAW_LIMIT;
        else
            velocity_to_body[0][1] = SIN_YAW_LIMIT;
    }
```

APPENDIX D - rwa_kinemat.c

```
    else
    {
*/
        velocity_to_body[0][0] = cos_yaw;
        velocity_to_body[0][1] = -sin_yaw;
/*
    }
*/
    velocity_to_body[0][2] =  0.0;
    velocity_to_body[1][0] =  -velocity_to_body[0][1] * temp;
    velocity_to_body[1][1] =  velocity_to_body[0][0] * temp;
    velocity_to_body[2][0] =  velocity_to_body[1][2] * velocity_to_body[0][1];
    velocity_to_body[2][1] = -velocity_to_body[1][2] * velocity_to_body[0][0];
    velocity_to_body[2][2] =  velocity_to_body[1][1] * velocity_to_body[0][0] -
            velocity_to_body[1][0] * velocity_to_body[0][1];
    ang_vel = vehicle_angular_velocity ();
    body_to_world = rotate_get_mat (hull (), world ());
    gravity[X] = body_to_world[0][2];
    gravity[Y] = body_to_world[1][2];
    gravity[Z] = body_to_world[2][2];
    g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
    vertical_speed = vec_dot_prod (norm_vel, gravity);
    if (true_airspeed >= DISPLAY_SPEED_LIMIT)
        velocity_pitch = asin (vertical_speed);
    else
        velocity_pitch = 0.0;
    vertical_speed *= true_airspeed;
    body_pitch = asin (body_to_world[1][2]);
    gravity[X] = -gravity[X];
    gravity[Y] = -gravity[Y];
    gravity[Z] = -gravity[Z];
    temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
            body_to_world[1][1] * body_to_world[1][1]);
    if (temp < E_NANO)
    {
        roll = 0.0;
        heading = 0.0;
    }
    else
    {
      temp2 = (body_to_world[0][0] * body_to_world[1][1] -
            body_to_world[0][1] * body_to_world[1][0]) / temp;
      if (temp2 > 1.0) temp2 = 1.0;
      roll = acos (temp2);
        if (body_to_world[1][1] * body_to_world[2][0] -
                body_to_world[1][0] * body_to_world[2][1] < 0.0)
            roll = -roll;
        if (body_to_world[1][0] >= 0.0)
            heading = acos (body_to_world[1][1] / temp);
        else
            heading = acos (-body_to_world[1][1] / temp) + PI;
    }
/*  NO METERS FOR NOW
    meter_g_force_set (g_force);
    meter_vertical_speed_set (vertical_speed);
```

APPENDIX D - rwa_kinemat.c

```
    if (true_airspeed >= DISPLAY_SPEED_LIMIT)
        meter_send_aero_data (rad_to_deg (body_pitch), rad_to_deg (roll),
                  ` rad_to_deg (heading), asin (sin_aoa), asin (sin_yaw),
                    indicated_airspeed, altitude, g_force);
    else
        meter_send_aero_data (0.0, 0.0,
                    rad_to_deg (heading), 0.0, 0.0,
                    indicated_airspeed, altitude, g_force);
*/
}

REAL kinematics_get_aoa ()
{
    return (asin (-velocity_to_body[1][2]));
}

REAL kinematics_get_yaw ()
{
    return (asin (-velocity_to_body[0][1]));
}

REAL kinematics_get_altitude ()
{
    return (altitude);
}

REAL kinematics_get_body_pitch ()
{
    return (body_pitch + body_pitch_offset);
}

REAL kinematics_get_velocity_pitch ()
{
    return (velocity_pitch);
}

REAL kinematics_get_roll ()
{
    return (roll);
}

REAL kinematics_get_heading ()
{
    return (heading);
}

REAL kinematics_get_true_airspeed ()
{
    return (true_airspeed);
}

REAL kinematics_get_indicated_airspeed ()
{
    return (indicated_airspeed);
}
```

APPENDIX D - rwa_kinemat.c

```
REAL kinematics_get_g_force ()
{
    return (g_force);
}

REAL kinematics_get_vertical_speed ()
{
    return (vertical_speed);
}

REAL *kinematics_get_gravity_vector ()
{
    return (gravity);
}

REAL *kinematics_get_linear_velocity_vector()
{
    return (velocity_vector);
}

REAL *kinematics_get_normalized_velocity_vector ()
{
    if (true_airspeed > DISPLAY_SPEED_LIMIT)
        return (norm_vel);
    else if (norm_vel[Y] >= 0.0)
        return (pos_unit_vel);
    else
        return (neg_unit_vel);
}

REAL *kinematics_get_angular_velocity_vector ()
{
    return (ang_vel);
}

T_MAT_PTR kinematics_get_velocity_to_body ()
{
    return (velocity_to_body);
}
```

Appendix E - Source code listing for miss_adat.c.

The following appendix contains the source code listing for miss_adat.c for convenience in document maintenance and understanding of the CSU.

APPENDIX E - miss_adat.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_ada
t.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_adat.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:12:09  cm-adst
 * R.Branson's changes for weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_adat.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/****************************************************************************
 *
 * Revisions:
 *
 *,    Version       Date      Author       Title                     SP/CR Number
 *     _______       ________  ________     ______________________    ____________
 *
 *     1.2           10/23/92  R. Branson   Data File Initiali-
 *                                            zation
 *     1.3           10/30/92  R. Branson   Added pathname to data
 *                                            directory
 *     1.4           11/25/92  R. Branson   Changed %i to %d
 *     1.5           01/19/93  P.Desmeules  Increased the size of the     31
 *                                            fgets to make sure the
 *                                            whole line is read in.
 *
****************************************************************************/

/****************************************************************************
 *
 *       SP/CR No.          Description of Modification
 *       ___________        ______________________________________________
 *
 *                          Hard coded defines changed to array elements.
 *                          Characteristics/parameter data array added.
 *                          Engine initialization data array added.
 *                          Degree of polynomial data array added.
 *                          Added file reads for ADAT characteristics/
 *                             parameters, burn speed coefficients, coast speed
 *                             coefficients, burn turn coefficients, coast turn
 *                             coefficients, and temporal bias coefficients.
 *
 *                          Added "/simnet/data/" to each data file pathname.
 *
****************************************************************************/

/*************************************************************
 *                                                           *
```

APPENDIX E - miss_adat.c

```
 * FILE:       miss_adat.c                                        *
 * AUTHOR:     Bryant Collard                                     *
 * MAINTAINER: Bryant Collard                                     *
 * PURPOSE:    This file contains routines which fly out a        *
 *             missile with the characteristics of a ADAT         *
 *             missile.                                           *
 * HISTORY:    06/28/89 bryant: Creation                          *
 *              08/06/90 bryant: NIU librva modifications.        *
 *                                                                *
 *                                                                *
 *                                                                *
 * Copyright (c) 1989 BBN Systems and Technologies, Inc.          *
 * All rights reserved.                                           *
 *                                                                *
 *****************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"

#include "miss_adat.h"

#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
 * Debug macro
 */
#ifdef FILEDBG
#define P(a)       a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define ADAT_BURNOUT_TIME      adat_miss_char[ 0]
#define ADAT_MAX_FLIGHT_TIME   adat_miss_char[ 1]
#define INVEST_DIST_SQ         adat_miss_char[ 2]
#define HELO_FUZE_DIST_SQ      adat_miss_char[ 3]
#define AIR_FUZE_DIST_SQ       adat_miss_char[ 4]
#define ADAT_TEMP_BIAS_TIME    adat_miss_char[ 5]
#define CLOSE_RANGE            adat_miss_char[ 6]

/*/
 *  Define the states the _ADAT_MISSILE_ can be in.
/*/
```

APPENDIX E - miss_adat.c

```
#define ADAT_FREE     0      /* No missile assigned. */
#define ADAT_GUIDE    1      /* Missile flying and guided. */
#define ADAT_UNGUIDE 2       /* Missile flying but unguided. */
#define ADAT_CLOSE    3      /* Missile flying against a close target. */
#define ADAT_HOT      4      /* Missile fired without cooling. */

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/

#define ADAT_BURN_SPEED_DEG    adat_miss_poly_deg[ 0]
#define ADAT_COAST_SPEED_DEG   adat_miss_poly_deg[ 1]
#define ADAT_BURN_TURN_DEG     adat_miss_poly_deg[ 2]
#define ADAT_COAST_TURN_DEG    adat_miss_poly_deg[ 3]
#define ADAT_TEMP_BIAS_DEG     adat_miss_poly_deg[ 4]

/*/
 *  ADAT missile characteristic parameters initialized to default values.
/*/

static REAL adat_miss_char[10] =
{
    48.0,     /* ticks (3.2 sec) */
   300.00,    /* ticks (20.0 sec) */
 90000.0,     /* (300 m) ** 2 */
    49.0,     /* (7 m) ** 2 */
   196.0,     /* (14 m) ** 2 */
    60.0,     /* ticks (4.0 sec) */
  2200.0,     /* close range*/
     0.0,
     0.0,
     0.0
};

/*/
 *  The following are the default values of the degree of polynomials.
/*/

static int adat_miss_poly_deg[5] =
{
   2,          /* Speed before motor burnout. */
   4,          /* Speed after motor burnout. */
   3,          /* Cosine of max turn before burnout. */
   5,          /* Cosine of max turn after burnout. */
   4           /* Temporial bias. */
};

/*/
 *  Coefficients for the speed polynomial before motor burnout.
/*/

static REAL adat_burn_speed_coeff[10] =
{
```

```
    2.296,                /* a_0 - m/tick */
    0.72990856,           /* a_1 - m/tick**2 */
    0.013310932,          /* a_2 - m/tick**3 */
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};

/*/
 *  Coefficients for the speed polynomial after motor burnout.
/*/

static REAL adat_coast_speed_coeff[10] =
{
   105.52162,             /* a_0 - m/tick */
    -1.0157285,           /* a_1 - m/tick**2 */
     5.6124330e-3,        /* a_2 - m/tick**3 */
    -1.6262608e-5,        /* a_3 - m/tick**4 */
     1.8991982e-8,        /* a_4 - m/tick**5 */
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
};

/*/
 *  Coefficients for the cosine of max turn polynomial before motor burnout.
/*/

static REAL adat_burn_turn_coeff[10] =
{
     0.999993,            /* a_0 - cos(rad)/tick */
    -6.2386917e-7,        /* a_1 - cos(rad)/tick**2 */
     1.6146426e-7,        /* a_2 - cos(rad)/tick**3 */
    -9.720142e-7,         /* a_3 - cos(rad)/tick**4 */
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
};

/*/
 *  Coefficients for the cosine of max turn polynomial after motor burnout.
/*/

static REAL adat_coast_turn_coeff[10] =
{
     0.99753111,          /* a_0 - cos(rad)/tick */
```

APPENDIX E - miss_adat.c

```
    5.5817986e-5,          /* a_1 - cos(rad)/tick**2 */
    -5.1276276e-7,         /* a_2 - cos(rad)/tick**3 */
    2.2388593e-9,          /* a_3 - cos(rad)/tick**4 */
    -5.1964622e-12,        /* a_4 - cos(rad)/tick**5 */
    4.5499104e-15,         /* a_5 - cos(rad)/tick**6 */
    0.0,
    0.0,
    0.0,
    0.0
};

/*/
 *  Coefficients for the temporial bias polynomial.
/*/

static REAL adat_temp_bias_coeff[10] =
{
    5.3105657e-2,          /* a_0 - m */
    7.1795817e-2,          /* a_1 - m/tick */
    1.8084646e-2,          /* a_2 - m/tick**2 */
   -6.0083762e-4,          /* a_3 - m/tick**3 */
    4.6761091e-6,          /* a_4 - m/tick**4 */
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};

/*/
 *  The following arrays are used to give the missile the proper superelevation
 *  at launch time.  Two are required to deal with launches off either side
 *  of the turret.
/*/

static T_MATRIX tube_C_sight_left;
static T_MATRIX tube_C_sight_right;

/*/
 *  Memory for the missiles is declared in vehicle specific code.  During
 *  initialization, a pointer is assigned to this memory then some memory
 *  issues are dealt with in this module.
/*/

static ADAT_MISSILE *adat_array;     /* A pointer to missile memory. */
static int num_adats;                /* The number of defined missiles. */

/*/
 *  Declare static functions.
/*/

/* static void missile_adat_fly (); ** made external */
static void missile_adat_stop ();
```

APPENDIX E - miss_adat.c

```
/****************************************************************
 *                                                              *
 * ROUTINE: missile_adat_init                                   *
 * PARAMETERS:    missile_array - A pointer to an array of      *
 *                          ADAT missiles defined in            *
 *                          vehicle specific code.              *
 *          num_missiles - The number missiles defined in       *
 *                          _missile_array_.                    *
 * RETURNS: none                                                *
 * PURPOSE: This routine copies the parameters into             *
 *          variables static to this module and initializes     *
 *          the state of all the missiles.  It also             *
 *          initializes the proximity fuze.                     *
 *                                                              *
 ****************************************************************/

void missile_adat_init (missile_array, num_missiles)
ADAT_MISSILE missile_array[];
int num_missiles;
{
      int   i;     /* A counter. */
      REAL  mag; /* Used to generate tube to sight matricies. */
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ ADAT missile file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR miss_adat.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_ad_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_ad_ch.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            adat_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("adat_miss_char(%3d) is%11.3f %s", i,
                  adat_miss_char[i], descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR miss_adat.c READ FROM FILE      */

/*  DEFAULT BURN SPEED DATA FOR miss_adat.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_ad_bs.d","r");
      if(fp==NULL){
```

APPENDIX E - miss_adat.c

```
        fprintf(stderr, "Cannot open /simnet/data/ms_ad_bs.d\n");
        exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    ADAT_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("adat_miss_poly_deg(0) is%3d %s",
          ADAT_BURN_SPEED_DEG, descript););

    /*    Read array data   */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          adat_burn_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("adat_burn_speed_coeff(%3d) is%11.3f %s", i,
                      adat_burn_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_adat.c READ FROM FILE      */

/*  DEFAULT COAST SPEED DATA FOR miss_adat.c READ FROM FILE         */
    fp = fopen("/simnet/data/ms_ad_cs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_ad_cs.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    ADAT_COAST_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("adat_miss_poly_deg(1) is%3d %s",
          ADAT_COAST_SPEED_DEG, descript););

    /*    Read array data   */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          adat_coast_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("adat_coast_speed_coeff(%3d) is%11.3f %s", i,
                      adat_coast_speed_coeff[i], descript););
          ++i;
    }
```

APPENDIX E - miss_adat.c

```
    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_adat.c READ FROM FILE    */

/*  DEFAULT BURN TURN DATA FOR miss_adat.c READ FROM FILE          */
    fp = fopen("/simnet/data/ms_ad_bt.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_ad_bt.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    ADAT_BURN_TURN_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("adat_miss_poly_deg(2) is%3d %s",
          ADAT_BURN_TURN_DEG, descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          adat_burn_turn_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("adat_burn_turn_coeff(%3d) is%11.3f %s", i,
                   adat_burn_turn_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN TURN DATA FOR miss_adat.c READ FROM FILE      */

/*  DEFAULT COAST TURN DATA FOR miss_adat.c READ FROM FILE         */
    fp = fopen("/simnet/data/ms_ad_ct.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_ad_ct.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    ADAT_COAST_TURN_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("adat_miss_poly_deg(3) is%3d %s",
          ADAT_COAST_TURN_DEG, descript););

    /*    Read array data    */
    i=0;
```

```
     while(fscanf(fp,"%f", &data_tmp) != EOF){
           adat_coast_turn_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("adat_coast_turn_coeff(%3d) is%11.3f %s", i,
                        adat_coast_turn_coeff[i], descript););
           ++i;
     }

     fclose(fp);
/*  END DEFAULT COAST TURN DATA FOR miss_adat.c READ FROM FILE      */

/*  DEFAULT TEMP BIAS DATA FOR miss_adat.c READ FROM FILE           */
     fp = fopen("/simnet/data/ms_ad_tb.d","r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_ad_tb.d\n");
           exit();
     }

     rewind(fp);

     /*    Read degree of polynomial */

     fscanf(fp,"%d", &data_tmp_int);
     ADAT_TEMP_BIAS_DEG = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("adat_miss_poly_deg(4) is%3d %s",
           ADAT_TEMP_BIAS_DEG, descript););

     /*    Read array data    */
     i=0;

     while(fscanf(fp,"%f", &data_tmp) != EOF){
           adat_temp_bias_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("adat_temp_bias_coeff(%3d) is%11.3f %s", i,
                        adat_temp_bias_coeff[i], descript););
           ++i;
     }

     fclose(fp);
/*  END DEFAULT TEMP BIAS DATA FOR miss_adat.c READ FROM FILE       */

    num_adats = num_missiles;
    adat_array = missile_array;
    for (i = 0; i < num_missiles; i++)
    {
        adat_array[i].mptr.state = ADAT_FREE;
        adat_array[i].mptr.max_flight_time = ADAT_MAX_FLIGHT_TIME;
        adat_array[i].mptr.max_turn_directions = 1;
    }
/**/
 *  Initialize the proximity fuze.
/**/
    missile_fuze_prox_init ();
/**/
```

APPENDIX E - miss_adat.c

```
 *  Initialize the tube to sight transformation matricies.
/*/
    mag = sqrt (adat_burn_speed_coeff[0] * adat_burn_speed_coeff[0] +
            2.0 * adat_temp_bias_coeff[0] * adat_temp_bias_coeff[0]);
    tube_C_sight_right[1][0] = adat_temp_bias_coeff[0] / mag;
    tube_C_sight_right[1][1] = adat_burn_speed_coeff[0] / mag;
    tube_C_sight_right[1][2] = adat_temp_bias_coeff[0] / mag;
    mag = sqrt (tube_C_sight_right[1][0] * tube_C_sight_right[1][0] +
            tube_C_sight_right[1][1] * tube_C_sight_right[1][1]);
    tube_C_sight_right[0][0] = tube_C_sight_right[1][1] / mag;
    tube_C_sight_right[0][1] = -tube_C_sight_right[1][0] / mag;
    tube_C_sight_right[0][2] = 0.0;
    tube_C_sight_right[2][0] = tube_C_sight_right[1][2] *
            tube_C_sight_right[0][1];
    tube_C_sight_right[2][1] = -tube_C_sight_right[1][2] *
            tube_C_sight_right[0][0];
    tube_C_sight_right[2][2] = mag;
    mat_copy (tube_C_sight_right, tube_C_sight_left);
    tube_C_sight_left[0][1] = -tube_C_sight_left[0][1];
    tube_C_sight_left[1][0] = -tube_C_sight_left[1][0];
    tube_C_sight_left[2][0] = -tube_C_sight_left[2][0];
}

int missile_adat_is_free( missile )
int missile;
{
    return( (adat_array[missile].mptr.state == ADAT_FREE ));
}


/****************************************************************
 *                                                    *
 * ROUTINE: missile_adat_fire                         *
 * PARAMETERS:    aptr - A pointer to the ADAT missile to be      *
 *                 fired.                             *
 *          target_type - The missile can be set for three  *
 *                        types of targets by the launching *
 *                        vehicle.  This variable stores    *
 *                        the setting.                      *
 *          launch_point - The location in world            *
 *                         coordinates that the missile is  *
 *                         launched from.                   *
 *          loc_sight_to_world - The sight to world         *
 *                               transformation matrix used *
 *                               only in this routine.      *
 *          launch_speed - The speed of the launch          *
 *                         platform (assumed to be in the   *
 *                         direction of the missile).       *
 *          range_to_intercept - Range to intercept.        *
 *          tube - The tube the missile was launched from.  *
 *          target_vehicle_id - The vehicle ID of the       *
 *                              target (if any).            *
 * RETURNS: TRUE if successful, FALSE if not.               *
 * PURPOSE: This routine performs the functions             *
 *          specifically related to the firing of a ADAT    *
```

APPENDIX E - miss_adat.c

```
*         missile.                                     *
*                                                      *
*****************************************************************/

int missile_adat_fire (aptr, target_type, launch_point, loc_sight_to_world,
        launch_speed, range_to_intercept, tube, target_vehicle_id)
ADAT_MISSILE *aptr;
int target_type;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
REAL range_to_intercept;
int tube;
VehicleID *target_vehicle_id;
{
    int i;                      /* A counter. */
    MISSILE *mptr;              /* Pointer to the particular generic missile
                                   pointed at by _aptr_. */
    int comm_target_type;       /* Indication of whether target is known. */
/*/
 *  Find _mptr_ and _target_id_.
/*/
    mptr = &(aptr->mptr);
    if (target_vehicle_id == 0)
        aptr->target_vehicle_id.vehicle = vehicleIrrelevant;
    else
        aptr->target_vehicle_id = *target_vehicle_id;
/*/
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
/*/
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    if (range_to_intercept < CLOSE_RANGE)
        mat_copy (loc_sight_to_world, mptr->orientation);
    else
    {
        if (((tube / 2) * 2) == tube)
            mat_mat_mul (tube_C_sight_left, loc_sight_to_world,
                    mptr->orientation);
        else
            mat_mat_mul(tube_C_sight_right, loc_sight_to_world,
                    mptr->orientation);
    }
    mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
            adat_burn_speed_coeff, 0.0) + launch_speed;
    mptr->init_speed = launch_speed;
/*/
 *  Indicate that the proximity fuze has no vehicles it is tracking.
/*/
    aptr->pptr = NULL;
/*/
 *  Set fuze distance and fuze target according to missile target
 *  setting.  Set network variables.
/*/
```

APPENDIX E - miss_adat.c

```
    switch (target_type)
    {
    case ADAT_TGT_GND:
        aptr->fuze_dist_sq = 0.0;
        aptr->target_flag = PROX_FUZE_ON_NO_VEH;
        break;
    case ADAT_TGT_HELO:
        aptr->fuze_dist_sq = HELO_FUZE_DIST_SQ;
        if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
            aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
        else
            aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
        break;
    case ADAT_TGT_AIR:
        aptr->fuze_dist_sq = AIR_FUZE_DIST_SQ;
        if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
            aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
        else
            aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
        break;
    default:
        aptr->fuze_dist_sq = 0.0;
        aptr->target_flag = PROX_FUZE_ON_NO_VEH;
        printf ("MISS_ADAT: Unknown target type %d\n", target_type);
        break;
    }
    if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
        comm_target_type = targetUnknown;
    else
        comm_target_type = targetIsVehicle;
/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, return FALSE.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_ADATS),
            munition_US_ADATS, munition_US_ADATS, &(aptr->target_vehicle_id),
            comm_target_type, objectIrrelevant, tube))
        return (FALSE);
/*/
 *  If all was successful, put any flying missiles in an unguided state
 *  and put this missile in a guided state.
/*/
    for (i = 0; i < num_adats; i++)
    {
        if ((adat_array[i].mptr.state == ADAT_GUIDE) ||
                (adat_array[i].mptr.state == ADAT_CLOSE))
            adat_array[i].mptr.state = ADAT_UNGUIDE;
    }
    if (range_to_intercept < CLOSE_RANGE)
        mptr->state = ADAT_CLOSE;
    else
        mptr->state = ADAT_GUIDE;
    return (TRUE);
}
```

APPENDIX E - miss_adat.c

```
/*****************************************************************
 *                                                              *
 * ROUTINE: missile_adat_fly_missiles                           *
 * PARAMETERS:    sight_location - The location in world        *
 *                             coordinates of the gunner's      *
 *                             sight.                           *
 *          loc_sight_to_world - The sight to world             *
 *                                transformation matrix used    *
 *                                only in this routine.         *
 *              veh_list - Vehicle list ID.                     *
 * RETURNS: none                                                *
 * PURPOSE: This routine flies out all missiles in a            *
 *          flying state.                                       *
 *                                                              *
 *****************************************************************/

void missile_adat_fly_missiles (sight_location, loc_sight_to_world, veh_list)
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
int veh_list;
{
    int i;      /* A counter. */
/*/
 *  Fly out all flying missiles.
/*/
    for (i = 0; i < num_adats; i++)
    {
        if (adat_array[i].mptr.state != ADAT_FREE)
            missile_adat_fly (&(adat_array[i]), sight_location,
                    loc_sight_to_world, i, veh_list);
    }
}


/*****************************************************************
 *                                                              *
 * ROUTINE: missile_adat_fly                                    *
 * PARAMETERS:    aptr - A pointer to the ADAT missile that is to *
 *                 be flown out.                                *
 *          sight_location - The location in world              *
 *                           coordinates of the gunner's        *
 *                           sight.                             *
 *          loc_sight_to_world - The sight to world             *
 *                               transformation matrix used     *
 *                               only in this routine.          *
 *          tube - The tube the missile was launched from.      *
 *              veh_list - Vehicle list ID.                     *
 * RETURNS: none                                                *
 * PURPOSE: This routine performs the functions                 *
 *          specifically related to the flying a ADAT           *
 *          missile.                                            *
 *                                                              *
 *****************************************************************/
```

APPENDIX E - miss_adat.c

```
void missile_adat_fly (aptr, sight_location, loc_sight_to_world, tube,
    veh_list)
ADAT_MISSILE *aptr;
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
int tube;
int veh_list;
{
    MISSILE *mptr;          /* A pointer to the generic aspects of _aptr_. */
    REAL time;              /* The current time after launch (ticks). */
    REAL bias;              /* The value of the temporal bias. */
/*/
 *  Set _mptr_ and _time_.  These values are created mostly for increased
 *  readablity.
/*/
    mptr = &(aptr->mptr);
    time = mptr->time;
/*/
 *  Find the current missile speed and the cosines of the maximum allowed turn
 *  angles in each direction.  The equations used are different before and
 *  after motor burnout.
/*/
    if (time < ADAT_BURNOUT_TIME)
    {
        mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
                adat_burn_speed_coeff, time) + mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (ADAT_BURN_TURN_DEG,
                adat_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
                adat_coast_speed_coeff, time) + mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (ADAT_COAST_TURN_DEG,
                adat_coast_turn_coeff, time);
    }
/*/
 *  Find the target point, etc.
/*/
    if ((mptr->state == ADAT_GUIDE) || (mptr->state == ADAT_CLOSE))
    {
        if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state == ADAT_GUIDE))
        {
            bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
                    adat_temp_bias_coeff, time);
            if (((tube / 2) * 2) == tube)
                missile_target_los_bias (mptr, sight_location,
                        loc_sight_to_world, -bias, bias);
            else
                missile_target_los_bias (mptr, sight_location,
                        loc_sight_to_world, bias, bias);
        }
        else
            missile_target_los (mptr, sight_location, loc_sight_to_world);
    }
```

```
    else if (mptr->state == ADAT_UNGUIDE)
        missile_target_unguided (mptr);
    else
        printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
/*/
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_adat_stop (aptr);
        return;
    }
    else
    {
/*/
 *      If the missile successfully flew, process the proximity fuze.
/*/
        missile_fuze_prox (mptr, MSL_TYPE_MISSILE, aptr->target_flag,
                &(aptr->target_vehicle_id), &(aptr->pptr), veh_list,
                INVEST_DIST_SQ, aptr->fuze_dist_sq);
/*/
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE))
        {
            missile_adat_stop (aptr);
            return;
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************************
 *                                                                           *
 * ROUTINE: missile_adat_reset_missiles                                      *
 * PARAMETERS:    none                                                       *
 * RETURNS: none                                                             *
 * PURPOSE: This routine puts any flying missile into an                     *
 *          unguided state.                                                  *
 *                                                                           *
 *****************************************************************************/

void missile_adat_reset_missiles ()
{
    int i;          /* A counter. */
/*/
 *  Reset all flying missiles.
/*/
    for (i = 0; i < num_ad..ts; i++)
```

```
    {
        if ((adat_array[i].mptr.state == ADAT_GUIDE) ||
                (adat_array[i].mptr.state == ADAT_CLOSE))
            adat_array[i].mptr.state = ADAT_UNGUIDE;
    }
}

/******************************************************************
 *                                                              *
 * ROUTINE: missile_adat_stop                                   *
 * PARAMETERS:    aptr - A pointer to the ADAT missile that is to *
 *                 be stopped.                                  *
 * RETURNS: none                                                *
 * PURPOSE: This routine causes all concerned to forget         *
 *          about the missile.  It should be called when        *
 *          the flyout of any ADAT missile is stopped *
 *          (whether or not it has exploded).  Note that        *
 *          this routine can only be called within this         *
 *          module.                                             *
 *                                                              *
 ******************************************************************/

static void missile_adat_stop (aptr)
ADAT_MISSILE *aptr;
{
/*/
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
/*/
    missile_fuze_prox_stop (&(aptr->pptr));
    missile_util_comm_stop_missile (&(aptr->mptr), MSL_TYPE_MISSILE);
    aptr->mptr.state = ADAT_FREE;
}
```

Appendix F - Source code listing for miss_atgm.c.

The following appendix contains the source code listing for miss_atgm.c for convenience in document maintenance and understanding of the CSU.

APPENDIX F - miss_atgm.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_atg
m.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_atgm.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:12:37  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_atgm.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/**************************************************************************
 *
 * Revisions:
 *
 *     Version        Date       Author        Title                      SP/CR Number
 *     _______      ________    _________     ______________________     ____________
 *
 *     1.2          10/23/92    R. Branson    Data File Initiali-
 *                                               zation
 *     1.3          10/30/92    R. Branson    Added pathname to data
 *                                               directory
 *     1.4          11/25/92    R. Branson    Changed %i to %d
 *
 *     1.5          01/19/93    P.Desmeules   Increased the size of the     31
 *                                               fgets to make sure the
 *                                             ..whole line is read.in.
 **************************************************************************/

/**************************************************************************
 *
 *       SP/CR No.        Description of Modification
 *       _________        ____________________________________________
 *
 *                        Hard coded defines changed to array elements.
 *                        Characteristics/parameter data array added.
 *                        Degree of polynomial data array added.
 *                        Added file reads for ATGM characteristics/
 *                           parameters, burn speed coefficients, coast speed
 *                           coefficients, burn turn coefficients, and coast
 *                           turn coefficients.
 *
 *                        Added "/simnet/data/" to each data file pathname.
 *
 **************************************************************************/

/************************************************************
 *                                                          *
 *                                                          *
 * FILE:      miss_atgm.c
```

APPENDIX F - miss_atgm.c

```
 * AUTHOR:      Bryant Collard                                         *
 * MAINTAINER:  Bryant Collard                                         *
 * PURPOSE:     This missile is the same as the tow except             *
 *                 it uses point targeting.  It flys to a point        *
 *                 rather than the view direction                      *
 *                                                                     *
 * HISTORY:     10/31/88 bryant: Creation                              *
 *               4/26/89 bryant: Added statically allocated mem        *
 *                                                                     *
 *                                                                     *
 *                                                                     *
 * Copyright (c) 1988 BBN Systems and Technologies, Inc.               *
 * All rights reserved.                                                *
 *                                                                     *
 **********************************************************************/

#include "stdio.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmatrix.h"
#include "libmap.h"
#include "librva.h"

#include "miss_atgm.h"

#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
 * Debug macro
 */
#ifdef FILEDBG
#define P(a)        a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define TOW_BURNOUT_TIME       tow_miss_char[0]
#define TOW_RANGE_LIMIT_TIME   tow_miss_char[1]
#define TOW_MAX_FLIGHT_TIME    tow_miss_char[2]
#define ATGM_TURN_FACTOR       tow_miss_char[3]

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/

#define TOW_BURN_SPEED_DEG  tow_miss_poly_deg[0]
```

APPENDIX F - miss_atgm.c

```
#define TOW_COAST_SPEED_DEG tow_miss_poly_deg[1]
#define TOW_BURN_TURN_DEG   tow_miss_poly_deg[2]
#define TOW_COAST_TURN_DEG  tow_miss_poly_deg[3]

/*/
 *  Tow missile characteristic parameters initialized to default values.
/*/
static REAL tow_miss_char[5] =
{
   24.0,     /* ticks (1.6 sec) */
  268.35,    /* ticks (17.89 sec) */
  200.00,    /* ticks - cos of max turn > 1.0 beyond this point */
    0.9,     /* ATGM turn factor for wider turning capability */
    0.0
};

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed and turn of the missile at any point in time.
/*/
static int tow_miss_poly_deg[5] =
{
   2,        /* Speed before motor burnout. */
   3,        /* Speed after motor burnout. */
   1,        /* Cosine of max turn before burnout. */
   3,        /* Cosine of max turn after burnout. */
   0         /* not used.  */
};

/*/
 *  Coefficients for the speed polynomial before motor burnout initialized to
 *  default values.
/*/

static REAL tow_burn_speed_coeff[5] =
{
    4.466666667,        /* a_0 - m/tick    ( 67.0 m/sec) */
    1.222103405,        /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
   -0.024532086,        /* a_2 - m/tick**3 (-82.7057910 m/sec**3) */
    0.0,
    0.0
};

/*/
 *  Coefficients for the speed polynomial after motor burnout initialized to
 *  default values.
/*/

static REAL tow_coast_speed_coeff[5] =
{
    21.81905383,        /* a_0 - m/tick    (327.2858074 m/sec) */
    -9.5382019e-2,      /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
     2.4378222e-4,      /* a_2 - m/tick**3 (  0.8227650 m/sec**3) */
    -2.6311111e-7,      /* a_3 - m/tick**4 ( -0.0133200 m/sec**4) */
     0.0
```

```
};

/*/
 *  Coefficients for the cosine of max turn polynomials before motor burnout.
 *  The structure _MAX_COS_COEFF_ is used to store the values for the turn
 *  sideways, up, and down polynomials along with their order.
/*/

static MAX_COS_COEFF tow_burn_turn_coeff =
{
    1,                      /* Order of the polynomials. */
    {
                            /* Sidewards turn. */
        0.999976868652,     /* a_0 - cos(rad)/tick */
       -3.5933955e-7        /* a_1 - cos(rad)/tick**2 */
    },
    {
                            /* Upwards turn. */
        0.999960667258,     /* a_0 - cos(rad)/tick */
       -3.1492328e-6        /* a_1 - cos(rad)/tick**2 */
    },
    {
                            /* Downwards turn. */
        0.999978909989,     /* a_0 - cos(rad)/tick */
       -7.8194991e-9        /* a_1 - cos(rad)/tick**2 */
    }
};

/*/
 *  Coefficients for the cosine of max turn polynomials after motor burnout.
/*/

static MAX_COS_COEFF tow_coast_turn_coeff =
{
    3,                      /* Order of the polynomials. */
    {
                            /* Sidewards turn. */
        0.99995112518,      /* a_0 - cos(rad)/tick */
        8.96333e-7,         /* a_1 - cos(rad)/tick**2 */
       -5.995375e-9,        /* a_2 - cos(rad)/tick**3 */
        1.162225e-11        /* a_3 - cos(rad)/tick**4 */
    },
    {
                            /* Upwards turn. */
        0.9998498495,       /* a_0 - cos(rad)/tick */
        1.657779e-6,        /* a_1 - cos(rad)/tick**2 */
       -8.231861e-9,        /* a_2 - cos(rad)/tick**3 */
        1.381832e-11        /* a_3 - cos(rad)/tick**4 */
    },
    {
                            /* Downwards turn. */
        0.9999714014,       /* a_0 - cos(rad)/tick */
        3.382077e-7,        /* a_1 - cos(rad)/tick**2 */
       -1.601259e-9,        /* a_2 - cos(rad)/tick**3 */
        2.623014e-12        /* a_3 - cos(rad)/tick**4 */
```

APPENDIX F - miss_atgm.c

```
    }
};

/*/
 *  Declare static functions.
/*/

static void missile_atgm_stop ();

/****************************************************************
 *                                                              *
 *                                                              *
 * ROUTINE: missile_atgm_init                                   *
 * PARAMETERS:    tptr - a pointer to the TOW to be             *
 *                  initialized.                                *
 * RETURNS: none                                                *
 * PURPOSE: This routine initializes the state of the           *
 *          missile to indicate that it is available and        *
 *          sets values that never change.                      *
 *                                                              *
 ****************************************************************/

void missile_atgm_init (tptr)
ATGM_MISSILE *tptr;
{
      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ ATGM missile file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR miss_atgm.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_at_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_at_ch.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            tow_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_miss_char(%3d) is%11.3f %s", i, tow_miss_char[i],
                        descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR miss_atgm.c READ FROM FILE     */
```

APPENDIX F - miss_atgm.c

```
/*  DEFAULT BURN SPEED DATA FOR miss_atgm.c READ FROM FILE          */
    fp = fopen("/simnet/data/ms_at_bs.d","r");
    if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_at_bs.d\n");
            exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    TOW_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("tow_miss_poly_deg(0) is%3d %s", TOW_BURN_SPEED_DEG,
                descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
            tow_burn_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_burn_speed_coeff(%3d) is%11.3f %s", i,
                            tow_burn_speed_coeff[i], descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_atgm.c READ FROM FILE      */

/*  DEFAULT COAST SPEED DATA FOR miss_atgm.c READ FROM FILE         */
    fp = fopen("/simnet/data/ms_at_cs.d","r");
    if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_at_cs.d\n");
            exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    TOW_COAST_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("tow_miss_poly_deg(1) is%3d %s", TOW_COAST_SPEED_DEG,
                descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
            tow_coast_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
```

APPENDIX F - miss_atgm.c

```
        P(printf("tow_coast_speed_coeff(%3d) is%11.3f %s", i,
                    tow_coast_speed_coeff[i], descript););
        ++i;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_atgm.c READ FROM FILE    */

/*  DEFAULT BURN TURN DATA FOR miss_atgm.c READ FROM FILE          */
    fp = fopen("/simnet/data/ms_at_bt.d","r");
    if(fp==NULL){
        fprintf(stderr, "Cannot open /simnet/data/ms_at_bt.d\n");
        exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    TOW_BURN_TURN_DEG = data_tmp_int;
    tow_burn_turn_coeff.deg = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("tow_miss_poly_deg(2) is%3d %s", TOW_BURN_TURN_DEG,
            descript););

    /*    Read array data    */

    for (i=0; i <= data_tmp_int; i++) {
        fscanf(fp,"%f", &data_tmp);
        tow_burn_turn_coeff.side_coeff[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("tow_burn_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                    tow_burn_turn_coeff.side_coeff[i], descript););
    }

    for (i=0; i <= data_tmp_int; i++) {
        fscanf(fp,"%f", &data_tmp);
        tow_burn_turn_coeff.up_coeff[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("tow_burn_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                    tow_burn_turn_coeff.up_coeff[i], descript););
    }

    for (i=0; i <= data_tmp_int; i++) {
        fscanf(fp,"%f", &data_tmp);
        tow_burn_turn_coeff.down_coeff[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("tow_burn_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                    tow_burn_turn_coeff.down_coeff[i], descript););
    }

    fclose(fp);
/*  END DEFAULT BURN TURN DATA FOR miss_atgm.c READ FROM FILE      */
```

APPENDIX F - miss_atgm.c

```
/*  DEFAULT COAST TURN DATA FOR miss_atgm.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_at_ct.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_at_ct.d\n");
            exit();
      }

      rewind(fp);

      /*    Read degree of polynomial */

      fscanf(fp,"%d", &data_tmp_int);
      TOW_COAST_TURN_DEG = data_tmp_int;
      tow_coast_turn_coeff.deg = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("tow_miss_poly_deg(3) is%3d %s", TOW_COAST_TURN_DEG,
                  descript););

      /*    Read array data    */

      for (i=0; i <= data_tmp_int; i++) {
            fscanf(fp,"%f", &data_tmp);
            tow_coast_turn_coeff.side_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.side_coeff[i], descript););
      }

      for (i=0; i <= data_tmp_int; i++) {
            fscanf(fp,"%f", &data_tmp);
            tow_coast_turn_coeff.up_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.up_coeff[i], descript););
      }

      for (i=0; i <= data_tmp_int; i++) {
            fscanf(fp,"%f", &data_tmp);
            tow_coast_turn_coeff.down_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.down_coeff[i], descript););
      }

      fclose(fp);
/*  END DEFAULT COAST TURN DATA FOR miss_atgm.c READ FROM FILE      */

    tptr->mptr.state = FALSE;
    tptr->mptr.max_flight_time = TOW_MAX_FLIGHT_TIME;
    tptr->mptr.max_turn_directions = 3;

    /*********************************************************************/
    /* change turn polynomial coefficients so missile has larger  */
    /* max turn angle.  Since Ph determines when a vehicle should be    */
    /* impacted, turn rates should not effect missile effectiveness     */
```

```
/*************************************************************************/
for (i=0; i<tow_burn_turn_coeff.deg; i++)
  {
  tow_burn_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
  tow_burn_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
  tow_burn_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
  }
for (i=0; i<tow_coast_turn_coeff.deg; i++)
  {
  tow_coast_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
  tow_coast_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
  tow_coast_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
  }
}


/****************************************************************
 *                                                              *
 * ROUTINE: missile_atgm_fire                                   *
 * PARAMETERS:    tptr - A pointer to the TOW missile to be     *
 *                  fired.                                      *
 * PARAMETERS:    launch_point - The location in world          *
 *                         coordinates that the missile is      *
 *                         launched from.                       *
 *          loc_sight_to_world - The sight to world             *
 *                              transformation matrix used      *
 *                              only in this routine.           *
 *          launch_speed - The speed of the launch              *
 *                         platform (assumed to be in the       *
 *                         direction of the missile).           *
 *          tube - The tube the missile was launched from.      *
 * RETURNS: none                                                *
 * PURPOSE: This routine performs the functions                 *
 *          specifically related to the firing of a TOW         *
 *          missile.                                            *
 *                                                              *
 ****************************************************************/

ATGM_MISSILE *missile_atgm_fire (tptr, launch_point, loc_sight_to_world,
        launch_speed, tube, try_to_hit_target, target_id, target_loc)
ATGM_MISSILE *tptr;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
int tube;
int try_to_hit_target;
VehicleID target_id;
VECTOR target_loc;
{
    MISSILE *mptr;          /* Pointer to the particular generic missile
                               pointed at by _tptr_. */

 /*/
  *  Find _mptr_.
 /*/
    mptr = &(tptr->mptr);
```

```
/*/
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
/*/
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (loc_sight_to_world, mptr->orientation);
    mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
            tow_burn_speed_coeff, 0.0) + launch_speed;
    mptr->init_speed = launch_speed;
/*/
 *  Set the wire as uncut.
/*/
    tptr->wire_is_cut = FALSE;
/*/
 *  if we are trying to hit a target then save the target_id.  Otherwise,
 *  save the target location (some point in space)
/*/
    tptr->try_to_hit_target = try_to_hit_target;
    if (try_to_hit_target)
      tptr->target_id = target_id;
    else
      {
      vec_copy(target_loc, tptr->target_location);
      }


/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_TOW),
            munition_US_TOW, munition_US_TOW, NULL, targetUnknown,
            objectIrrelevant, tube))
        return;
/*/
 *  If all was successful, set the missile state to TRUE and return.
/*/
    mptr->state = TRUE;
    return;
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_atgm_fly                                     *
 * PARAMETERS:    tptr - A pointer to the TOW missile that is to  *
 *                 be flown out.                                 *
 *           sight_location - The location in world              *
 *                            coordinates of the gunner's        *
 *                            sight.                             *
 *           loc_sight_to_world - The sight to world             *
 *                            transformation matrix used         *
 *                            only in this routine.              *
 * RETURNS: none                                                 *
```

APPENDIX F - miss_atgm.c

```
 * PURPOSE: This routine performs the functions        *
 *          specifically related to the flying a TOW   *
 *          missile.                                *
 *                                                  *
 **************************************************************/

void missile_atgm_fly (tptr, sight_location, loc_sight_to_world)
ATGM_MISSILE *tptr;
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
{
    MISSILE *mptr;          /* A pointer to the generic aspects of _tptr_. */
    REAL time;              /* The current time after launch (ticks). */
    VehicleAppearanceVariant *target_vehicle;
                    /* pointer to target vehicles appearance packet */

    VECTOR target_plus_offset;  /* this vector gives a targets location
                            with     an appropriate offset for ground
                            vehs     */
    static VECTOR ground_veh_offset = {0.0, 0.0, 1.0};
                        /* offset to aim missile at for ground vehs */
/*/
 *  Set _mptr_ and _time_.  These values are created mostly for increased
 *  readablity.
/*/
    mptr = &(tptr->mptr);
    time = mptr->time;
/*/
 *  If the missile has reached its maximum range (not the maximum distance
 *  its allowed to fly), cut the wire.
/*/
    if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)
        tptr->wire_is_cut = TRUE;
/*/
 *  Find the current missile speed and the cosines of the maximum allowed turn
 *  angles in each direction.  The equations used are different before and
 *  after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
    {
        mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                tow_burn_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                tow_coast_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
    }
/*/
 *  If the wire has been cut, set the ground as the target; otherwise,
 *  find a target point which will fly the missile along the gunner's line of
 *  sight.  This targeting scheme takes into account the errors introduced by
 *  attempting to guide the missile in a canted position.
```

```
/*/
    if (tptr->wire_is_cut)
      {
      printf("G");
        missile_target_ground (mptr);
      }
    else
      {
      /*     if operator has successfully designated a target then
       * try_to_hit_target will be true.  Therefore, we search the
       * list of targets for the vehicleID and fly missile to that
       * location.
       *     if try_to_hit_target is false then target point is passed
       * and we should fly the missile to the target_point.
       *     if try_to_hit_target is true and we can't find the
       * vehicle id in the rva list then the vehicle has dropped off the
       * net and we fly the missile into the ground.
       *
       */
      if (tptr->try_to_hit_target)
          {
          if ((target_vehicle = rva_get_veh_app_pkt (&(tptr->target_id))) !=
            NULL)
            {
            /*****************************************************/
            /* if the target is a ground vehicle we need to guide */
            /* the missile to a point other than the center of mass     */
            /* for SIMNET ground vehicles the center of mass is on      */
            /* the ground.  This causes missiles to fly into the  */
            /* ground                                            */
            /*****************************************************/
            if ((target_vehicle->guises.distinguished &
                 (objectDomainMask | vehicleEnvironmentMask)) ==
                 (objectDomainVehicle | vehicleEnvironmentGround))
                 {
                 vec_add (target_vehicle->location, ground_veh_offset,
                   target_plus_offset);
                 }
            else
                 {
                 vec_copy (target_vehicle->location, target_plus_offset);
                 }

            missile_target_point(mptr, target_plus_offset);
            }
          else
            {
            /* printf("g"); */
            missile_target_unguided (mptr);
            }
          }
      else
          {
          /* printf("p"); */
          /*************************************************************/
```

```
        /* guide the missile toward a point for 5 ticks, then just     */
        /* fly it straight ahead.  With the wide turning radius */
        /* missile will fly around in circles otherwise          */
        /*************************************************************/
        if (time < 5.0)
            missile_target_point(mptr, tptr->target_location);
        else
            missile_target_unguided (mptr);
        }
    }
/*/
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_atgm_stop (tptr);
        return;
    }
    else
    {
/*/
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
        {
            missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
            missile_atgm_stop (tptr);
            return;
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************
 *                                                  *
 * ROUTINE: missile_atgm_stop                       *
 * PARAMETERS:    tptr - A pointer to the TOW missile that is to   *
 *                 be stopped.                           *
 * RETURNS: none                                    *
 * PURPOSE: This routine causes all concerned to forget        *
 *          about the missile.  It should be called when       *
 *          the flyout of any TOW missile is stopped  *
 *          (whether or not it has exploded).  Note that       *
 *          this routine can only be called within this        *
 *          module.                                  *
 *                                                  *
 *****************************************************************/

static void missile_atgm_stop (tptr)
```

APPENDIX F - miss_atgm.c

```
ATGM_MISSILE *tptr;
{
/*/
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
/*/
    missile_util_comm_stop_missile (&(tptr->mptr), MSL_TYPE_MISSILE);
    tptr->mptr.state = FALSE;
}

/****************************************************************
 *                                                    *
 * ROUTINE: missile_atgm_cut_wire                             *
 * PARAMETERS:    tptr - A pointer to the TOW missile whose wire    *
 *           is to be cut.                                    *
 * RETURNS: none                                          *
 * PURPOSE: This routine sets a flag indicating that the       *
 *           guidance wire of this missile is cut.            *
 *                                                    *
 ****************************************************************/

void missile_atgm_cut_wire (tptr)
ATGM_MISSILE *tptr;
{
/*/
 *  If the the wire is not already cut, cut the wire.
/*/
    if (!tptr->wire_is_cut)
        tptr->wire_is_cut = TRUE;
}
```

Appendix G - Source code listing for miss_hellfr.c.

The following appendix contains the source code listing for miss_atgm.c for convenience in document maintenance and understanding of the CSU.

APPENDIX G - miss_hellfr.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_hel
lfr.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_hellfr.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  00:45:01  cm-adst
 * R. Branson's weapon changes.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_hellfr.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/*****************************************************************************
 *
 * Revisions:
 *
 *    Version        Date        Author       Title                    SP/CR Number
 *    _______        ________    _________    ______________________   ____________
 *
 *     1.2          10/23/92    R. Branson   Data File Initiali-
 *                                              zation
 *     1.3          10/30/92    R. Branson   Added pathname to data
 *                                              directory
 *     1.4          11/25/92    R. Branson   Changed %i to %d
 *
 *     1.5          01/19/93    P.Desmeules  Increased the size of the       31
 *                                              fgets to make sure the
 *                                              whole line is read in.
 *
 *****************************************************************************/

/*****************************************************************************
 *
 *        SP/CR No.        Description of Modification
 *        _________        ____________________________________________________
 *
 *
 *                         Hard coded defines changed to array elements.
 *                         Characteristics/parameter data array added.
 *                         Degree of polynomial data array added.
 *                         Added file reads for hellfire characteristics/
 *                            parameters, burn speed coefficients, coast speed
 *                            coefficients, and time-of-flight coefficients.
 *
 *                         Added "/simnet/data/" to each data file pathname.
 *
 *****************************************************************************/

/*************************************************************
 *                                                           *
 *                                                           *
 * FILE:        miss_hellfr.c                                *
 * AUTHOR:      Bryant Collard
```

APPENDIX G - miss_hellfr.c

```
 * MAINTAINER: Bryant Collard                                        *
 * PURPOSE:    This file contains routines which fly out a           *
 *             missile with the characteristics of a HELLFIRE        *
 *             missile.                                              *
 * HISTORY:    11/25/88 bryant: Creation                             *
 *              4/24/89 bryant: Added static memory allocation       *
 *              08/07/90 bryant: NIU librva modifications.           *
 *             08/09/90 kris: corrected flight coefficients          *
 *                                                                   *
 * Copyright (c) 1988 BBN Systems and Technologies, Inc.             *
 * All rights reserved.                                              *
 *                                                                   *
 *********************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmatrix.h"
#include "libmap.h"
/*-- need Range_Squared info --*/
#include "libhull.h"
#include "libkin.h"
/*----------------------------*/

#include "miss_hellfr.h"
#include "libmissile.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define HELLFIRE_ARM_TIME           hellfr_miss_char[ 0]
#define HELLFIRE_BURNOUT_TIME       hellfr_miss_char[ 1]
#define HELLFIRE_MAX_FLIGHT_TIME    hellfr_miss_char[ 2]
#define SPEED_0                     hellfr_miss_char[ 3]
#define THETA_0                     hellfr_miss_char[ 4]
/*/
 *  Set parameters which will control flight trajectory behavior.
/*/
```

APPENDIX G - miss_hellfr.c

```
#define SIN_UNGUIDE                  hellfr_miss_char[ 5]
#define COS_UNGUIDE                  hellfr_miss_char[ 6]
#define SIN_CLIMB                    hellfr_miss_char[ 7]
#define COS_CLIMB                    hellfr_miss_char[ 8]
#define SIN_LOCK                     hellfr_miss_char[ 9]
#define COS_LOCK                     hellfr_miss_char[10]
#define COS_TERM                     hellfr_miss_char[11]
#define COS_LOSE                     hellfr_miss_char[12]

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/
#define HELLFIRE_TOF_DEG             hellfr_miss_poly_deg[ 0]
#define HELLFIRE_BURN_SPEED_DEG      hellfr_miss_poly_deg[ 1]
#define HELLFIRE_COAST_SPEED_DEG     hellfr_miss_poly_deg[ 2]

/*/
 *  Hellfire missile characteristic parameters initialized to default values.
/*/
static REAL hellfr_miss_char[15] =
{
   20.0,              /* ticks (1.3 sec) */
   36.0,              /* ticks (2.4 sec) */
  540.0,              /* ticks (36 sec)  */
   30.95953043,       /* max_speed       */
    0.046542113,
    0.069756474,      /* sin 4.0 deg     */
    0.997564050,      /* cos 4.0 deg     */
    0.004072424,      /* sin 3.5 deg     */
    0.999991708,      /* cos 3.5 deg     */
    0.156434465,      /* sin 9.0 deg     */
    0.987688341,      /* cos 9.0 deg     */
    0.241921896,      /* cos 76.0 deg    */
    0.939692621,      /* cos 20.0 deg    */
    0.0,
    0.0
};

/*/
 *  Hellfire missile polynomial degree initialized to default values.
/*/
static int hellfr_miss_poly_deg[ 3] =
{
    4,    /* tof poly degree         */
    3,    /* burn speed poly degree  */
    5     /* coast speed poly degree */
};

/*/
 *  Coefficients for the TOF polynomial initialized to default values.
/*/
static REAL hellfire_tof_coeff[10] =
{
```

APPENDIX G - miss_hellfr.c

```
    18.0,              /* a_0  tick          */ /* 1.2 seconds */
    3.1461816e-2,      /* a_1  tick/meter    */
    3.1921274e-6,      /* a_2  tick/meter^2 */
    3.5260413e-10,     /* a_3  tick/meter^3 */
   -2.8469594e-14,     /* a_4  tick/meter^4 */
    0.0,               /* a_5  tick/meter^5 */
    0.0,               /* a_6  tick/meter^6 */
    0.0,               /* a_7  tick/meter^7 */
    0.0,               /* a_8  tick/meter^8 */
    0.0                /* a_9  tick/meter^9 */
};

/*/
 *  Coefficients for the speed polynomial before motor burnout initialized to
 *  default values.
/*/
static REAL hellfire_burn_speed_coeff[10] =
{
     2.0044395e-2,      /* a_0 - meters   */
     6.7384206e-1,      /* a_1 - m/tick   */
     9.8007701e-3,      /* a_2 - m/tick^2 */
    -1.6782227e-4,      /* a_3 - m/tick^3 */
     0.0,               /* a_4 - m/tick^4 */
     0.0,               /* a_5 - m/tick^5 */
     0.0,               /* a_6 - m/tick^6 */
     0.0,               /* a_7 - m/tick^7 */
     0.0,               /* a_8 - m/tick^8 */
     0.0                /* a_9 - m/tick^9 */
};

/*/
 *  Coefficients for the speed polynomial after motor burnout initialized to
 *  default values.
/*/
static REAL hellfire_coast_speed_coeff[10] =
{
     4.2738447e+1,      /* a_0 - meters   */
    -4.1048613e-1,      /* a_1 - m/tick   */
     2.6023604e-3,      /* a_2 - m/tick^2 */
    -8.4870417e-6,      /* a_3 - m/tick^3 */
     1.3322932e-8,      /* a_4 - m/tick^4 */
    -7.9542005e-12,     /* a_5 - m/tick^5 */
     0.0,               /* a_6 - m/tick^6 */
     0.0,               /* a_7 - m/tick^7 */
     0.0,               /* a_8 - m/tick^8 */
     0.0                /* a_9 - m/tick^9 */
};

static ObjectType hellfire_ammo_type = munition_US_Hellfire;
static REAL
    max_range_limit,     /* [ MISSILE_US_MAX_RANGE_LIMIT ]            */
    max_range_squared,   /* [ MISSILE_US_MAX_RANGE_LIMIT ^ 2 ]        */
    speed_factor;        /* [ MISSILE_US_SPEED_FACTOR ]               */

/*/
```

APPENDIX G - miss_hellfr.c

```
 *  Declare static functions.
/*/
static void missile_hellfire_stop ();

/*****************************************************************
 *                                                      *
 * ROUTINE: missile_hellfire_init                              *
 * PARAMETERS:    mptr - a pointer to the HELLFIRE to be              *
 *                 initialized.                                *
 * RETURNS: none                                        *
 * PURPOSE: This routine initializes the state of the        *
 *          missile to indicate that it is available and     *
 *          sets values that never change.                   *
 *                                                      *
 *****************************************************************/

void missile_hellfire_init (mptr)
MISSILE *mptr;
{
      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ HELLFIRE missile file data $$$$\n"););

 /*  DEFAULT CHARACTERISTIC DATA FOR miss_hellfr.c READ FROM FILE  */
      fp = fopen("/simnet/data/ms_hf_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_ch.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF)
        {
            hellfr_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("hellfr_miss_char(%3d) is%11.3f %s", i,
                        hellfr_miss_char[i], descript););
            ++i;
      }

      fclose(fp);
 /*  END DEFAULT CHARACTERISTIC DATA FOR miss_hellfr.c READ FROM FILE     */

 /*  DEFAULT TIME-OF-FLIGHT DATA FOR miss_hellfr.c READ FROM FILE  */
      fp = fopen("/simnet/data/ms_hf_tf.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_tf.d\n");
```

```
        exit();
    }

    rewind(fp);

    /*    Read degree of polynomial      */

      fscanf(fp,"%d", &data_tmp_int);
    hellfr_miss_poly_deg[0] = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("hellfr_miss_poly_deg(0) is%3d %s",
                        hellfr_miss_poly_deg[0], descript););

    /*    Read array data    */

    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF)
      {
            hellfire_tof_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("hellfire_tof_coeff(%3d) is%11.3f %s", i,
                        hellfire_tof_coeff[i], descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT TIME-OF-FLIGHT DATA FOR miss_hellfr.c READ FROM FILE     */

/*  DEFAULT BURN SPEED DATA FOR miss_hellfr.c READ FROM FILE  */
    fp = fopen("/simnet/data/ms_hf_bs.d","r");
    if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_bs.d\n");
            exit();
    }

    rewind(fp);

    /*    Read degree of polynomial      */

      fscanf(fp,"%d", &data_tmp_int);
    hellfr_miss_poly_deg[1] = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("hellfr_miss_poly_deg(1) is%3d %s",
                        hellfr_miss_poly_deg[1], descript););

    /*    Read array data    */

    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF)
      {
            hellfire_burn_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("hellfire_burn_speed_coeff(%3d) is%11.3f %s", i,
```

APPENDIX G - miss_hellfr.c

```
                    hellfire_burn_speed_coeff[i], descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_hellfr.c READ FROM FILE  */

/*  DEFAULT COAST SPEED DATA FOR miss_hellfr.c READ FROM FILE  */
    fp = fopen("/simnet/data/ms_hf_cs.d","r");
    if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_cs.d\n");
            exit();
    }

    rewind(fp);

    /*    Read degree of polynomial    */

      fscanf(fp,"%d", &data_tmp_int);
    hellfr_miss_poly_deg[2] = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("hellfr_miss_poly_deg(2) is%3d %s",
                    hellfr_miss_poly_deg[2], descript););

    /*    Read array data    */

    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF)
      {
            hellfire_coast_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("hellfire_coast_speed_coeff(%3d) is%11.3f %s", i,
                    hellfire_coast_speed_coeff[i], descript););
            ++i;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_hellfr.c READ FROM FILE */

    mptr->state = FALSE;
    mptr->max_flight_time = HELLFIRE_MAX_FLIGHT_TIME;
    mptr->max_turn_directions = 1;
    speed_factor = MISSILE_US_SPEED_FACTOR;
    max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
    max_range_squared = max_range_limit * max_range_limit;
    hellfire_ammo_type = munition_US_Hellfire;
}

void missile_hellfire_set_speed_factor( scale_speed )
REAL scale_speed;
{
    speed_factor = scale_speed;
}
```

APPENDIX G - miss_hellfr.c

```
void missile_hellfire_set_max_range_limit( limit_range )
REAL limit_range;
{
    max_range_limit = limit_range;
    max_range_squared = max_range_limit * max_range_limit;
}

void missile_hellfire_set_ammo_type( ammo )
ObjectType ammo;
{
    hellfire_ammo_type = ammo;
}

/****************************************************************
 * ROUTINE: missile_hellfire_calc_tof                          *
 * PARAMETERS:    range - Range to target.                        *
 * RETURNS: Time Of Flight for _range_ meters to target.     *
 * PURPOSE: This routine evaluates the TOF poly and returns *
 *              the time of flight for a Hellfire Missile        *
 *              to fly _range_ meters.                            *
 ****************************************************************/

REAL missile_hellfire_calc_tof( range )
REAL range;
{
    REAL time;
    time =
        missile_util_eval_poly( HELLFIRE_TOF_DEG, hellfire_tof_coeff, range );
    return( (time / speed_factor) );
}

/****************************************************************
 *                                                      *
 * ROUTINE: missile_hellfire_fire                              *
 * PARAMETERS:    mptr - A pointer to the HELLFIRE missile that    *
 *                  is to be launched.                      *
 *          launch_point - The location in world               *
 *                          coordinates that the missile is   *
 *                          launched from.                  *
 *          launch_to_world - The transformation matrix of   *
 *                              the launch platform to the    *
 *                              world.                      *
 *          launch_speed - The speed of the launch             *
 *                          platform (assumed to be in the    *
 *                          direction of the missile). *
 *          tube - The tube the missile was launched from.    *
 * RETURNS: none                                        *
 * PURPOSE: This routine performs the functions             *
 *          specifically related to the firing of a            *
 *          Hellfire missile.                           *
 *                                                      *
 ****************************************************************/

void missile_hellfire_fire (mptr, launch_point, launch_to_world, launch_speed,
        tube)
```

APPENDIX G - miss_hellfr.c

```
MISSILE *mptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
{
/*/
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
/*/
#ifdef notdeff
    if( max_range_limit > 0.0 )
        mptr->max_flight_time =
            1.0 + missile_hellfire_calc_tof( max_range_limit );
#endif
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (launch_to_world, mptr->orientation);
    mptr->speed = launch_speed +
        (speed_factor * (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                                                 hellfire_burn_speed_coeff,
                                                 0.0) ));
    mptr->init_speed = launch_speed;
/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (hellfire_ammo_type),
            hellfire_ammo_type, hellfire_ammo_type, NULL,
          targetUnknown, objectIrrelevant, tube))
        return;
/*/
 *  If all was successful, set the missile state to TRUE and return.
/*/
    mptr->state = TRUE;
    return;
}

/*****************************************************************
 *                                                    *
 * ROUTINE: missile_hellfire_fly                           *
 * PARAMETERS:    mptr - A pointer to the HELLFIRE missile that   *
 *                 is to be flown out.                     *
 *          target_location - The location in world            *
 *                          coordinates of the target.         *
 * RETURNS: none                                      *
 * PURPOSE: This routine performs the functions            *
 *          specifically related to the flying a HELLFIRE      *
 *          missile.                                  *
 *                                                    *
 *****************************************************************/

void missile_hellfire_fly (mptr, target_location)
MISSILE *mptr;
```

```
VECTOR target_location;
{
    register REAL time;          /* The current time after launch (ticks). */
/*/
 *  Set and _time_.  This is created mostly for increased readablity.
/*/
    time = mptr->time;
/*/
 *  Find the current missile speed and the cosines of the maximum allowed turn
 *  angles in each direction.  The equations used are different before and
 *  after motor burnout.
/*/
    if (time < HELLFIRE_BURNOUT_TIME)
    {
        mptr->speed = mptr->init_speed +
             (speed_factor *
              (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                                       hellfire_burn_speed_coeff, time) ));
    }
    else
    {
        mptr->speed = mptr->init_speed +
             (speed_factor *
              (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                                       hellfire_coast_speed_coeff, time) ));
    }
/*/
 *  Note that this is a temporary method of finding the max turn angle.
/*/
    mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / SPEED_0) * THETA_0);
/*/
 *  If the missile is not armed, fly in a search trajectory; otherwise, fly
 *  in a targeted trajectory.
/*/
    if( max_range_limit > 0  &&
       kinematics_range_squared (veh_kinematics, mptr->location) >
       max_range_squared )
        missile_target_ground( mptr );
    else if (time < HELLFIRE_ARM_TIME)
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB,
             COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
    else
        missile_target_agm (mptr, target_location, SIN_UNGUIDE, COS_UNGUIDE,
             SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
/*/
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_hellfire_stop (mptr);
        return;
    }
    else
    {
```

```
/*/
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
        {
            missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
            missile_hellfire_stop (mptr);
            return;
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_hellfire_stop                               *
 * PARAMETERS:    mptr - A pointer to the HELLFIRE missile that *
 *                  is to be stopped.                           *
 * RETURNS: none                                                *
 * PURPOSE: This routine causes all concerned to forget         *
 *          about the missile.  It should be called when        *
 *          the flyout of any HELLFIRE missile is stopped       *
 *          (whether or not it has exploded).  Note that        *
 *          this routine can only be called within this         *
 *          module.                                             *
 *                                                              *
 *****************************************************************/

static void missile_hellfire_stop (mptr)
MISSILE *mptr;
{
/*/
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
/*/
    missile_util_comm_stop_missile (mptr, MSL_TYPE_MISSILE);
    mptr->state = FALSE;
}
```

Appendix H - Source code listing for miss_kem.c.

The following appendix contains the source code listing for miss_kem.c for convenience in document maintenance and understanding of the CSU.

APPENDIX H - miss_kem.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_kem
.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_kem.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:13:01  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_kem.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/*****************************************************************************
 *
 * Revisions:
 *
 *      Version        Date       Author      Title                   SP/CR Number
 *      _______       ________   __________  ____________________    ____________
 *
 *      1.2            10/23/92   R. Branson  Data File Initiali-
 *                                              zation
 *      1.3            10/30/92   R. Branson  Added pathname to data
 *                                              directory
 *      1.4            11/25/92   R. Branson  Changed %i to %d
 *
 *      1.5            01/19/93   P.Desmeules Increased the size of the      31
 *                                              fgets to make sure the
 *                                              whole line is read in.
 *****************************************************************************/

/*****************************************************************************
 *
 *      SP/CR No.      Description of Modification
 *      _________      ____________________________________________________
 *
 *                     Hard coded defines changed to array elements.
 *                     Characteristics/parameter data array added.
 *                     Degree of polynomial data array added.
 *                     Added file reads for KEM characteristics/parameters,
 *                        burn speed coefficients, coast speed coefficients,
 *                        burn turn coefficients, and coast turn coeffi-
 *                        cients.
 *
 *                     Added "/simnet/data/" to each data file pathname.
 *
 *****************************************************************************/

/*****************************************************************
 *                                                               *
 * FILE:        miss_kem.c                                       *
```

APPENDIX H - miss_kem.c

```
                                                                   *
 * AUTHOR:      Kris Bartol                                          *
 * MAINTAINER:  Kris Bartol: converted from miss_adat                *
 *                                                                   *
 * PURPOSE:     This file contains routines which fly out a          *
 *              missile with the characteristics of a KEM            *
 *              missile.                                             *
 * HISTORY:     10/23/90 kris: converted from miss_adat              *
 *                                                                   *
 * Copyright (c) 1989 BBN Systems and Technologies, Inc.             *
 * All rights reserved.                                              *
 *                                                                   *
 *******************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"

#include "miss_kem.h"

#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)       a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define KEM_BURNOUT_TIME        kem_miss_char[0]
#define KEM_MAX_FLIGHT_TIME     kem_miss_char[1]
/*
 * just after burnout, max V = ~3418 m/tick = ~230 m/sec
 * so in order to get the KEM missile to fly @ Vmax = 1524 m/2
 * must multiply the speed calculated by 6.626 ~= 1524 / 230
 */
#define KEM_TO_MACH5_FACTOR     kem_miss_char[2]

/*/
 *  Define the states the _KEM_MISSILE_ can be in.
/*/

#define KEM_FREE     0       /* No missile assigned. */
```

APPENDIX H - miss_kem.c

```
#define KEM_GUIDE   1      /* Missile flying and guided. */
#define KEM_UNGUIDE 2      /* Missile flying but unguided. */

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/

#define KEM_BURN_SPEED_DEG    kem_miss_poly_deg[0]
#define KEM_COAST_SPEED_DEG   kem_miss_poly_deg[1]
#define KEM_BURN_TURN_DEG     kem_miss_poly_deg[2]
#define KEM_COAST_TURN_DEG    kem_miss_poly_deg[3]

/*/
 *  ADAT missile characteristic parameters initialized to default values.
/*/

static REAL kem_miss_char[10] =
{
    48.0,    /* ticks (3.2 sec) */
   300.00,   /* ticks (20.0 sec) */
     6.626,  /* speed factor to raise from ADAT to KEM */
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
};

/*/
 *  The following are the default values of the degree of polynomials.
/*/

static int kem_miss_poly_deg[5] =
{
   2,          /* Speed before motor burnout. */
   4,          /* Speed after motor burnout. */
   3,          /* Cosine of max turn before burnout. */
   5,          /* Cosine of max turn after burnout. */
   0
};

/*/
 *  Coefficients for the speed polynomial before motor burnout initialized
 *  to default values.
/*/

static REAL kem_burn_speed_coeff[10] =
{
    2.296,              /* a_0 - m/tick */
    0.72990856,         /* a_1 - m/tick**2 */
    0.013310932,        /* a_2 - m/tick**3 */
```

```
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};

/*/
 *  Coefficients for the speed polynomial after motor burnout.
/*/

static REAL kem_coast_speed_coeff[10] =
{
  105.52162,              /* a_0 - m/tick */
   -1.0157285,            /* a_1 - m/tick**2 */
    5.6124330e-3,         /* a_2 - m/tick**3 */
   -1.6262608e-5,         /* a_3 - m/tick**4 */
    1.8991982e-8,         /* a_4 - m/tick**5 */
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};

/*/
 *  Coefficients for the cosine of max turn polynomial before motor burnout.
/*/

static REAL kem_burn_turn_coeff[10] =
{
    0.999993,             /* a_0 - cos(rad)/tick */
   -6.2386917e-7,         /* a_1 - cos(rad)/tick**2 */
    1.6146426e-7,         /* a_2 - cos(rad)/tick**3 */
   -9.720142e-7,          /* a_3 - cos(rad)/tick**4 */
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};

/*/
 *  Coefficients for the cosine of max turn polynomial after motor burnout.
/*/

static REAL kem_coast_turn_coeff[10] =
{
    0.99753111,           /* a_0 - cos(rad)/tick */
    5.5817986e-5,         /* a_1 - cos(rad)/tick**2 */
   -5.1276276e-7,         /* a_2 - cos(rad)/tick**3 */
    2.2388593e-9,         /* a_3 - cos(rad)/tick**4 */
```

APPENDIX H - miss_kem.c

```
    -5.1964622e-12,         /* a_4 - cos(rad)/tick**5 */
     4.5499104e-15,         /* a_5 - cos(rad)/tick**6 */
     0.0,
     0.0,
     0.0,
     0.0
};

/*/
 *  Memory for the missiles is declared in vehicle specific code.  During
 *  initialization, a pointer is assigned to this memory then some memory
 *  issues are dealt with in this module.
/*/

static KEM_MISSILE *kem_array;      /* A pointer to missile memory. */
static int num_kems;                /* The number of defined missiles. */

/*/
 *  Declare static functions.
/*/

static void missile_kem_stop ();

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_kem_init                                    *
 * PARAMETERS:    missile_array - A pointer to an array of      *
 *                          KEM missiles defined in             *
 *                          vehicle specific code.              *
 *          num_missiles - The number missiles defined in       *
 *                         _missile_array_.                     *
 * RETURNS: none                                                *
 * PURPOSE: This routine copies the parameters into             *
 *          variables static to this module and initializes     *
 *          the state of all the missiles.                      *
 *                                                              *
 *****************************************************************/

void missile_kem_init (missile_array, num_missiles)
KEM_MISSILE missile_array[];
int num_missiles;
{
    int i;    /* A counter. */
      int  data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$$ KEM missile file data $$$$\n"););

 /*  DEFAULT CHARACTERISTICS DATA FOR miss_kem.c READ FROM FILE              */
      fp = fopen("/simnet/data/ms_km_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_km_ch.d\n");
            exit();
```

```
    }

    rewind(fp);

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          kem_miss_char[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kem_miss_char(%3d) is%11.3f %s", i,
                kem_miss_char[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR miss_kem.c READ FROM FILE       */

/*  DEFAULT BURN SPEED DATA FOR miss_kem.c READ FROM FILE            */
    fp = fopen("/simnet/data/ms_km_bs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_km_bs.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    KEM_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("kem_miss_poly_deg(0) is%3d %s",
          KEM_BURN_SPEED_DEG, descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          kem_burn_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kem_burn_speed_coeff(%3d) is%11.3f %s", i,
                      kem_burn_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_kem.c READ FROM FILE       */

/*  DEFAULT COAST SPEED DATA FOR miss_kem.c READ FROM FILE           */
    fp = fopen("/simnet/data/ms_km_cs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_km_cs.d\n");
          exit();
    }
```

```
    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    KEM_COAST_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("kem_miss_poly_deg(1) is%3d %s",
          KEM_COAST_SPEED_DEG, descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          kem_coast_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kem_coast_speed_coeff(%3d) is%11.3f %s", i,
                      kem_coast_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_kem.c READ FROM FILE     */

/*  DEFAULT BURN TURN DATA FOR miss_kem.c READ FROM FILE           */
    fp = fopen("/simnet/data/ms_km_bt.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_km_bt.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    KEM_BURN_TURN_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("kem_miss_poly_deg(2) is%3d %s",
          KEM_BURN_TURN_DEG, descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          kem_burn_turn_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kem_burn_turn_coeff(%3d) is%11.3f %s", i,
                      kem_burn_turn_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN TURN DATA FOR miss_kem.c READ FROM FILE       */
```

APPENDIX H - miss_kem.c

```
/*  DEFAULT COAST TURN DATA FOR miss_kem.c READ FROM FILE          */
     fp = fopen("/simnet/data/ms_km_ct.d","r");
     if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_km_ct.d\n");
          exit();
     }

     rewind(fp);

     /*    Read degree of polynomial */

     fscanf(fp,"%d", &data_tmp_int);
     KEM_COAST_TURN_DEG = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("kem_miss_poly_deg(3) is%3d %s",
          KEM_COAST_TURN_DEG, descript););

     /*    Read array data   */
     i=0;

     while(fscanf(fp,"%f", &data_tmp) != EOF){
          kem_coast_turn_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("kem_coast_turn_coeff(%3d) is%11.3f %s", i,
                    kem_coast_turn_coeff[i], descript););
          ++i;
     }

     fclose(fp);
/*  END DEFAULT COAST TURN DATA FOR miss_kem.c READ FROM FILE      */

    num_kems = num_missiles;
    kem_array = missile_array;
    for (i = 0; i < num_missiles; i++)
    {
        kem_array[i].mptr.state = KEM_FREE;
        kem_array[i].mptr.max_flight_time = KEM_MAX_FLIGHT_TIME;
        kem_array[i].mptr.max_turn_directions = 1;
    }
}

int missile_kem_is_free( missile )
int missile;
{
    return( (kem_array[missile].mptr.state == KEM_FREE ));
}

/*****************************************************************
 *                                                       *
 * ROUTINE: missile_kem_fire                             *
 * PARAMETERS:    kptr - A pointer to the KEM missile to be *
 *                fired.                                   *
 *          launch_point - The location in world           *
 *                         coordinates that the missile is  *
```

```
 *                      launched from.                 *
 *         loc_sight_to_world - The sight to world      *
 *                           transformation matrix used *
 *                           only in this routine.      *
 *         launch_speed - The speed of the launch       *
 *                      platform (assumed to be in the  *
 *                      direction of the missile).      *
 *             target_id - Target's tracking ID         *
 *             target_loc - location of target in World Coord *
 *         target_vehicle_id - The vehicle ID of the    *
 *                           target (if any).           *
 * RETURNS: TRUE if successful, FALSE if not.           *
 * PURPOSE: This routine performs the functions         *
 *          specifically related to the firing of a KEM *
 *          missile.                                    *
 *                                                      *
 ******************************************************************/

int missile_kem_fire (kptr, launch_point, loc_sight_to_world, launch_speed,
                      target_id, target_loc, target_vehicle_id)

KEM_MISSILE *kptr;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
int target_id;
VECTOR target_loc;
VehicleID *target_vehicle_id;
{
    int i;                      /* A counter. */
    MISSILE *mptr;              /* Pointer to the particular generic missile
                                   pointed at by _kptr_. */
    int comm_target_type;       /* Indication of whether target is known. */
/*
 *  Find _mptr_ and _target_id_.
 */
    mptr = &(kptr->mptr);
    if (target_vehicle_id == 0)
        kptr->target_vehicle_id.vehicle = vehicleIrrelevant;
    else
        kptr->target_vehicle_id = *target_vehicle_id;
    kptr->target_id = target_id;
    vec_copy( target_loc, kptr->target_pos );

/*
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
 */
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (loc_sight_to_world, mptr->orientation);

    mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) + launch_speed;
    mptr->init_speed = launch_speed;
```

```
    if (kptr->target_vehicle_id.vehicle == vehicleIrrelevant)
        comm_target_type = targetUnknown;
    else
        comm_target_type = targetIsVehicle;
/*
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, return FALSE.
 */
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_ADATS),
            munition_US_ADATS, munition_US_ADATS, &(kptr->target_vehicle_id),
            comm_target_type, objectIrrelevant, 0 /*tube*/))
        return (FALSE);
/*
 *  If all was successful, fly missile in guided state.
 */
    mptr->state = KEM_GUIDE;
    return (TRUE);
}

/****************************************************************
 *                                                              *
 * ROUTINE: missile_kem_update_guidance                         *
 * PARAMETERS:    missile - An index to the KEM missile that    *
 *                  is to be updated.                           *
 *           target_location - The location in world            *
 *                             coordinates of the target        *
 * RETURNS: none                                                *
 * PURPOSE: This routine updates the KEM's target's             *
 *           position in world coordinates.                     *
 *                                                              *
 ****************************************************************/

void missile_kem_update_guidance( missile, target_location )
int missile;
VECTOR target_location;
{
    if( kem_array[missile].mptr.state == KEM_GUIDE )
        vec_copy( target_location, kem_array[missile].target_pos );
}

/****************************************************************
 *                                                              *
 * ROUTINE: missile_kem_fly                                     *
 * PARAMETERS:    missile - An index to the KEM missile that    *
 *                  is to be flown out.                         *
 * RETURNS: none                                                *
 * PURPOSE: This routine performs the functions                 *
 *           specifically related to the flying a KEM           *
 *           missile.                                           *
 *                                                              *
 ****************************************************************/

void missile_kem_fly( missile )
int missile;
```

APPENDIX H - miss_kem.c

```
{
    KEM_MISSILE *kptr;      /* A pointer to a KEM missile  */
    MISSILE *mptr;          /* A pointer to the generic aspects of _kptr_. */
    REAL time;              /* The current time after launch (ticks). */
/*
 *  Set _kptr_, _mptr_ and _time_.  These values are created mostly
 *  for increased readablity.
 */
    kptr = &kem_array[missile];
    mptr = &(kptr->mptr);
    time = mptr->time;
/*
 *  Find the current missile speed and the cosines of the maximum allowed turn
 *  angles in each direction.  The equations used are different before and
 *  after motor burnout.
 */
    if (time < KEM_BURNOUT_TIME)
    {
        mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
                kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
                    mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (KEM_BURN_TURN_DEG,
                kem_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
                kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
                    mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (KEM_COAST_TURN_DEG,
                kem_coast_turn_coeff, time);
    }
/*
 *  Find the target point = Missile's Target's position regardless of state
 */
    if( mptr->state == KEM_GUIDE  ||  mptr->state == KEM_UNGUIDE )
        missile_target_point( mptr, kptr->target_pos );
    else
        printf ("MISSILE_KEM: disallowed missile state %d\n", mptr->state);
/*
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
 */
    if (!missile_util_flyout (mptr))  /* checks for time > max_flight_time */
    {
        missile_kem_stop (kptr);
        return;
    }
    else
    {
/*
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
 */
```

APPENDIX H - miss_kem.c

```
        if (missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE))
        {
            missile_kem_stop (kptr);
            return;
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_kem_reset_missiles                          *
 * PARAMETERS:    none                                          *
 * RETURNS: none                                                *
 * PURPOSE: This routine puts any flying missile into an        *
 *          unguided state.                                     *
 *                                                              *
 *****************************************************************/

void missile_kem_reset_missiles ()
{
    int i;
/*
 *  Reset all flying missiles.
 */
    for (i = 0; i < num_kems; i++)
        if( kem_array[i].mptr.state == KEM_GUIDE )
            kem_array[i].mptr.state = KEM_UNGUIDE;
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_kem_stop                                    *
 * PARAMETERS:    kptr - A pointer to the KEM missile that is to *
 *                  be stopped.                                 *
 * RETURNS: none                                                *
 * PURPOSE: This routine causes all concerned to forget         *
 *          about the missile.  It should be called when        *
 *          the flyout of any KEM missile is stopped            *
 *          (whether or not it has exploded).  Note that        *
 *          this routine can only be called within this         *
 *          module.                                             *
 *                                                              *
 *****************************************************************/

static void missile_kem_stop (kptr)
KEM_MISSILE *kptr;
{
/*/
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
/*/
```

```
    missile_util_comm_stop_missile (&(kptr->mptr), MSL_TYPE_MISSILE);
    kptr->mptr.state = KEM_FREE;
}
```

Appendix I - Source code listing for miss_maverck.c.

The following appendix contains the source code listing for miss_maverck.c for convenience in document maintenance and understanding of the CSU.

APPENDIX I - miss_maverck.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_mav
erck.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_maverck.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:13:31  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_maverck.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/*************************************************************************
 *
 * Revisions:
 *
 *     Version         Date        Author       Title                    SP/CR Number
 *     -------         --------    ----------   --------------------     ------------
 *
 *     1.2             10/23/92    R. Branson   Data File Initiali-
 *                                                zation
 *     1.3             10/30/92    R. Branson   Added pathname to data
 *                                                directory
 *     1.4             11/25/92    R. Branson   Changed %i to %d
 *
 *     1.5             01/19/93    P.Desmeules  Increased the size of the     31
 *                                                fgets to make sure the
 *                                                whole line is read in.
 *************************************************************************/

/*************************************************************************
 *
 *        SP/CR No.        Description of Modification
 *        ---------        -----------------------------------------------
 *
 *                         Hard coded defines changed to array elements.
 *                         Characteristics/parameter data array added.
 *                         Degree of polynomial data array added.
 *                         Added file reads for maverick characteristics/
 *                            parameters, burn speed coefficients, and coast
 *                            speed coefficients.
 *
 *                         Added "/simnet/data/" to each data file pathname.
 *
 *************************************************************************/

/****************************************************************
 *                                                              *
 * FILE:       miss_maverick.c                                  *
 * AUTHOR:     Bryant Collard                                   *
```

APPENDIX I - miss_maverck.c

```
 * MAINTAINER: Bryant Collard                                        *
 * PURPOSE:    This file contains routines which fly out a           *
 *             missile with the characteristics of a MAVERICK        *
 *             missile.                                              *
 * HISTORY:    12/8/88 bryant: Creation                              *
 *             4/24/89 bryant: Added static memory allocation.       *
 *             7/26/91 carol : libtrack/intervis integration         *
 *                                                                   *
 * Copyright (c) 1988 BBN Systems and Technologies, Inc.             *
 * All rights reserved.                                              *
 *                                                                   *
 ********************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libnear.h"
#include "libtrack.h"

#include "miss_maverck.h"

#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define MAVERICK_ARM_TIME           maverick_miss_char[ 0]
#define MAVERICK_BURNOUT_TIME       maverick_miss_char[ 1]
#define MAVERICK_MAX_FLIGHT_TIME    maverick_miss_char[ 2]
#define MAVERICK_LOCK_THRESHOLD     maverick_miss_char[ 3]
#define MAVERICK_HOLD_THRESHOLD     maverick_miss_char[ 4]
#define SPEED_0                   maverick_miss_char[ 5]
#define THETA_0                   maverick_miss_char[ 6]

/*/
 *  Set parmeters which will control flight trajectory behavior.
/*/
```

APPENDIX I - miss_maverck.c

```
#define SIN_UNGUIDE            maverick_miss_char[ 7]
#define COS_UNGUIDE            maverick_miss_char[ 8]
#define SIN_CLIMB              maverick_miss_char[ 9]
#define COS_CLIMB              maverick_miss_char[10]
#define SIN_LOCK               maverick_miss_char[11]
#define COS_LOCK               maverick_miss_char[12]
#define COS_TERM               maverick_miss_char[13]
#define COS_LOSE               maverick_miss_char[14]

/*/
 *  Define the states the _MAVERICK_MISSILE_ can be in.
/*/

#define MAVERICK_FREE    0      /* No missile assigned. */
#define MAVERICK_READY   1      /* Missile assigned to ready state. */
#define MAVERICK_FLYING  2      /* Missile assigned to flying state. */

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/

#define MAVERICK_BURN_SPEED_DEG  maverick_miss_poly_deg[0]
#define MAVERICK_COAST_SPEED_DEG maverick_miss_poly_deg[1]

/*/
 *  Maverick missile characteristic parameters initialized to default values.
/*/
static REAL maverick_miss_char[15] =
{
   20.0,          /* maverick arm time ticks (1.3 sec) */
   22.5,          /* maverick burnout time ticks (1.5 sec) */
  900.0,          /* maverick max flight time ticks (60 sec) */
    0.989073800,  /* maverick lock threshold cos (6 deg) ** 2 */
    0.969846310,  /* maverick hold threshold cos (10 deg) ** 2 */
   28.33333333,   /* speed_0      */
    0.046542113,  /* theta_0      */
    0.0,          /* sin level unguided flight. */
    1.0,      /* cos level unguided flight. */
    0.004072424,  /* sin climb 3.5 deg/sec */
    0.999991708,  /* cos climb 3.5 deg/sec */
    0.087155743,  /* sin lock 5 deg */
    0.996194698,  /* cos lock 5 deg */
    0.173648178,  /* cos terminal 80 deg */
    0.939692621   /* cos loose lock 20 deg */
};

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed.
/*/
static int maverick_miss_poly_deg[2] =
{
   1,        /* Maverick burn speed degree. */
```

APPENDIX I - miss_maverck.c

```
   3         /* Maverick coast speed degree. */
};

/*/
 *  Coefficients for the speed polynomial before motor burnout.
/*/

static REAL maverick_burn_speed_coeff[5] =
{
    0.03333333,          /* a_0 - m/tick (67.0 m/sec) */
    1.25777777           /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
};

/*/
 *  Coefficients for the speed polynomial after motor burnout.
/*/

static REAL maverick_coast_speed_coeff[5] =
{
    30.46972849,          /* a_0 - m/tick    (327.2858074 m/sec) */
  , -9.7721160e-2,        /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
     1.2433925e-4,        /* a_2 - m/tick**3 (0.8227650 m/sec**3) */
    -5.4061501e-8         /* a_3 - m/tick**4 (-0.0133200 m/sec**4) */
};

/*/
 *  Memory for the missiles is declared in vehicle specific code.  During
 *  initialization, a pointer is assigned to this memory then all memory
 *  issues are dealt with in this module.
/*/

static MAVERICK_MISSILE *maverick_array;  /* A pointer to missile memory. */
static int num_mavericks;                 /* The number of defined missiles. */

#define STRING_LEN  20
static char prelaunch_intervis_method [STRING_LEN + 1] = "lrf";
static char in_flight_intervis_method [STRING_LEN + 1] = "omniscient";
static PFI pel_callback_func;
static REAL maverick_cone_threshold;

/*/
 *  Declare static functions.
/*/

static void missile_maverick_fly ();
static MAVERICK_MISSILE *missile_maverick_get_missile_from_sensor_id ();
static void missile_maverick_lock_handler ();
static void missile_maverick_break_lock_handler ();
static REAL missile_maverick_detectibility ();
static void missile_maverick_object_update ();

/****************************************************************
 *                                                    *
 * ROUTINE: missile_maverick_init                     *
 * PARAMETERS:    missile_array - A pointer to an array of   *
```

APPENDIX I - miss_maverck.c

```
 *                          MAVERICK missiles defined in    *
 *                          vehicle specific code.          *
 *          num_missiles - The number missiles defined in   *
 *                         _missile_array_.                 *
 *                                                          *
 * RETURNS: none                                            *
 * PURPOSE: This routine copies the parameters into         *
 *          variables static to this module and initializes *
 *          the state of all the missiles.                  *
 *                                                          *
 ************************************************************/

void missile_maverick_init (missile_array, num_missiles, func)
MAVERICK_MISSILE missile_array[];
int num_missiles;
PFI func;
{
    int i;    /* A counter. */
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ MAVERICK missile file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR miss_maverck.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_mk_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_mk_ch.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data   */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            maverick_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("maverick_miss_char(%3d) is%11.3f %s", i,
                  maverick_miss_char[i], descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR miss_maverck.c READ FROM FILE   */

/*  DEFAULT BURN SPEED DATA FOR miss_maverck.c READ FROM FILE            */
      fp = fopen("/simnet/data/ms_mk_bs.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_mk_bs.d\n");
            exit();
      }

      rewind(fp);
```

APPENDIX I - miss_maverck.c

```
    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    MAVERICK_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("maverick_miss_poly_deg(0) is%3d %s",
          MAVERICK_BURN_SPEED_DEG, descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          maverick_burn_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("maverick_burn_speed_coeff(%3d) is%11.3f %s", i,
                       maverick_burn_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_maverck.c READ FROM FILE */

/*  DEFAULT COAST SPEED DATA FOR miss_maverck.c READ FROM FILE            */
    fp = fopen("/simnet/data/ms_mk_cs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_mk_cs.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    MAVERICK_COAST_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("maverick_miss_poly_deg(1) is%3d %s",
          MAVERICK_COAST_SPEED_DEG, descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          maverick_coast_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("maverick_coast_speed_coeff(%3d) is%11.3f %s", i,
                       maverick_coast_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_maverck.c READ FROM FILE        */

    maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
```

```
    num_mavericks = num_missiles;
    maverick_array = missile_array;

    for (i = 0; i < num_missiles; i++)
    {
      maverick_array[i].mptr.state = MAVERICK_FREE;
        maverick_array[i].mptr.max_flight_time = MAVERICK_MAX_FLIGHT_TIME;
        maverick_array[i].mptr.max_turn_directions = 1;
      maverick_array[i].object_being_tracked = NO_OBJECT;
      maverick_array[i].sensor_id = NULL;
    }
    pel_callback_func = func;
}

/*****************************************************************
 *                                                    *
 * ROUTINE: missile_maverick_sensor_init                    *
 * PARAMETERS:    none                                      *
 * RETURNS: none                                                  *
 * PURPOSE: Calls to initialize a libtrack sensor                 *
 *                                                    *
 *****************************************************************/
void missile_maverick_sensor_init (mvptr, iv_method)
MAVERICK_MISSILE *mvptr;
char *iv_method;
{
    if (TrackSensorInit (missile_maverick_lock_handler,
                   missile_maverick_break_lock_handler,
                   missile_maverick_detectibility,
                   pel_callback_func,
                   missile_maverick_object_update,
                   E_NANO,
                   &mvptr -> sensor_id) < 0)
        printf ("missile_maverick_sensor_init: TrackSensorInit: %s\n",
            TrackErrString ());

    if (TrackSetIntervisibility (mvptr -> sensor_id, prelaunch_intervis_method) <
0)
        printf ("missile_maverick_sensor_init: TrackSetIntervisibility: %s\n",
            TrackErrString ());

    if (TrackSetPersistence (mvptr -> sensor_id, 5 /* ticks of persistence */)
      < 0)
        printf ("missile_maverick_sensor_init: TrackSetPersistence: %s\n",
            TrackErrString ());

    if (TrackSetMaxResponses (mvptr -> sensor_id, 1) < 0)
        printf ("missile_maverick_sensor_init: TrackSetMaxResponses: %s\n",
            TrackErrString ());

    if (TrackSetVehicleID (mvptr -> sensor_id, network_get_vehicle_id ()) < 0)
        printf ("missile_maverick_sensor_init: TrackSetVehicleID: %s\n",
            TrackErrString ());
 }
```

APPENDIX I - miss_maverck.c

```
/****************************************************************
 *                                                    *
 * ROUTINE: missile_maverick_ready                          *
 * PARAMETERS:    none                                      *
 * RETURNS: A pointer to a missile that is currently  *
 *          available.                                *
 * PURPOSE: This routine finds, if possible, a missile that *
 *          is not being used, puts it in a ready state and *
 *          returns a pointer to it.                        *
 *                                                    *
 ****************************************************************/

MAVERICK_MISSILE *missile_maverick_ready ()
{
    int i;      /* A counter. */
/*/
 *  Try to find a free missile.
/*/
    for (i = 0; i < num_mavericks; i++)
    {
/*/
 *      If a free missile is found, put it in a ready state, clear the target
 *      ID and return a pointer to it.
/*/
        if (maverick_array[i].mptr.state == MAVERICK_FREE)
        {
            maverick_array[i].mptr.state = MAVERICK_READY;
            maverick_array[i].target_vehicle_id.vehicle = vehicleIrrelevant;
          missile_maverick_sensor_init (&maverick_array[i],
                              prelaunch_intervis_method);
            return (&maverick_array[i]);
        }
    }
/*/
 *  If no free missile is found, return a NULL pointer.
/*/
    return (NULL);
}


/****************************************************************
 *                                                    *
 * ROUTINE: missile_maverick_pre_launch                     *
 * PARAMETERS:    mvptr - A pointer to the missile that is to be   *
 *                   serviced.                        *
 *          launch_point - The location of the missile in    *
 *                         world coordinates.              *
 *          launch_to_world - The transformation matrix of  *
 *                            the missile to the world.    *
 *              veh_list - Vehicle list ID.                       *
 * RETURNS: none                                        *
 * PURPOSE: This routine is called after a missile has been *
 *          readied and before it has been launched.  It    *
 *          determines if the seeker head can see a target  *
 *          and, if it can see a target, stores its         *
```

```
 *          position.                         *
 *                                            *
 ************************************************************/
void missile_maverick_pre_launch (mvptr, launch_point, launch_to_world,
    veh_list)
MAVERICK_MISSILE *mvptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
int veh_list;
{
    register TObjectP object;
    VECTOR object_loc;
/*/
 *  tick libtrack to update location and see if any callbacks need to be
 *  invoked.
/*/
    if (TrackUpdate (mvptr -> sensor_id, veh_list, launch_point,
                launch_to_world[1]) < 0)
        printf ("missile_maverick_pre_launch: TrackUpdate: %s\n",
            TrackErrString ());
/*/
 *  If a target is found, store its location.
/*/
    if ((object = mvptr -> object_being_tracked) != NO_OBJECT)
    {
        mvptr->target_vehicle_id =  object -> var.vehicleID;
      GetLocationOfTObject (object, object_loc);
/* change pursuit to take a VECTOR rather than VAP for location */
        missile_target_pursuit (&(mvptr->mptr), object_loc);
    }
    else
    {
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
      if (TrackAcquire (mvptr -> sensor_id, veh_list, launch_point,
                launch_to_world[1]) < 0)
          printf ("missile_maverick_pre_launch: TrackAcquire: %s\n",
                TrackErrString ());
    }
}


/**************************************************************
 *                                                     *
 * ROUTINE: missile_maverick_fire                         *
 * PARAMETERS:    mvptr - A pointer to the MAVERICK missile that  *
 *                   is to be launched.                    *
 *          launch_point - The location in world              *
 *                       coordinates that the missile is     *
 *                       launched from.                  *
 *          launch_to_world - The transformation matrix of   *
 *                          the launch platform to the    *
 *                          world.                     *
 *          launch_speed - The speed of the launch            *
 *                       platform (assumed to be in the     *
 *                       direction of the missile). *
```

APPENDIX I - miss_maverck.c

```
 *          tube - The tube the missile was launched from.  *
 * RETURNS: TRUE for a successful launch and FALSE for an   *
 *          unsuccessful launch.
 * PURPOSE: This routine performs the functions          *
 *          specifically related to the firing of a         *
 *          MAVERICK missile.                     *
 *                                                *
 *****************************************************************/

int missile_maverick_fire (mvptr, launch_point, launch_to_world, launch_speed,
        tube)
MAVERICK_MISSILE *mvptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
{
    MISSILE *mptr;          /* Pointer to the particular generic missile
                               pointed at by _mvptr_. */
/*/
 *  Get a pointer to the generic elements of the MAVERICK missile.  This
 *  improves code readability.
/*/
    mptr = &(mvptr->mptr);
/*/
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
/*/
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (launch_to_world, mptr->orientation);
    mptr->speed = missile_util_eval_poly (MAVERICK_BURN_SPEED_DEG,
            maverick_burn_speed_coeff, 0.0) + launch_speed;
    mptr->init_speed = launch_speed;
/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, release the missile memory and return FALSE.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_Maverick),
            munition_US_Maverick, munition_US_Maverick,
            &(mvptr->target_vehicle_id), targetIsVehicle, objectIrrelevant,
            tube))
    {
        mptr->state = MAVERICK_FREE;
        return (FALSE);
    }
/*/
 *  If all was successful, set the missile state to MAVERICK_FLYING and
 *  return TRUE.
/*/
    mptr->state = MAVERICK_FLYING;
    return (TRUE);
}
```

APPENDIX I - miss_maverck.c

```
/*****************************************************************
 *                                                              *
 * ROUTINE: missile_maverick_fly_missiles                       *
 * PARAMETERS:    veh_list - Vehicle list ID.                   *
 * RETURNS: none                                                *
 * PURPOSE: This routine flies out all missiles in a            *
 *          flying state.                                       *
 *                                                              *
 *****************************************************************/

void missile_maverick_fly_missiles (veh_list)
int veh_list;
{
    int i;       /* A counter. */
/**/
 *  Fly out all flying missiles.
/**/
    for (i = 0; i < num_mavericks; i++)
    {
        if (maverick_array[i].mptr.state == MAVERICK_FLYING)
            missile_maverick_fly (&(maverick_array[i]), veh_list);
    }
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_maverick_fly                                *
 * PARAMETERS:    mvptr - A pointer to the MAVERICK missile that *
 *                   is to be flown out.                        *
 *               veh_list - Vehicle list ID.                    *
 * RETURNS: none                                                *
 * PURPOSE: This routine performs the functions                 *
 *          specifically related to the flying a MAVERICK       *
 *          missile.                                            *
 *                                                              *
 *****************************************************************/

static void missile_maverick_fly (mvptr, veh_list)
MAVERICK_MISSILE *mvptr;
int veh_list;
{
    register MISSILE *mptr;        /* A pointer to the generic aspects of
                                      _mvptr_. */
    REAL time;                     /* The current time after launch (ticks). */
    VECTOR target_location;        /* The location of the target. */
/**/
 *  Set _mptr_ and _time_.  These values are created mostly for increased
 *  readablity.
/**/
    mptr = &(mvptr->mptr);
    time = mptr->time;
/**/
 *  Find the current missile speed and the cosine of the maximum allowed turn
 *  angle.  The equations used are different before and after motor burnout.
```

APPENDIX I - miss_maverck.c

```
/*/
    if (time < MAVERICK_BURNOUT_TIME)
    {
        mptr->speed = missile_util_eval_poly (MAVERICK_BURN_SPEED_DEG,
                maverick_burn_speed_coeff, time) + mptr->init_speed;
    }
    else
    {
        mptr->speed = missile_util_eval_poly (MAVERICK_COAST_SPEED_DEG,
                maverick_coast_speed_coeff, time) + mptr->init_speed;
    }
/*/
 *  Note that this is a temporary method of finding turn angle.
/*/
    mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
            mptr->init_speed)) * THETA_0);

    if (TrackUpdate (mvptr -> sensor_id, veh_list, mptr -> location,
                mptr -> orientation[1]) < 0)
        printf ("missile_maverick_fly: TrackUpdate: %s\n", TrackErrString ());
/*/
 *  Find the target point to which the missile is to fly.  The missile ignores
 *  any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB,
                COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
    else
    {
      TObjectP object = mvptr -> object_being_tracked;
/*/
 *      Try to find a target.  If one is found, fly towards it in the
 *      proper trajectory, otherwise, fly in a search trajectory.
/*/
        if (object != NO_OBJECT)
        {
          VECTOR target_location;
          GetLocationOfTObject (object, target_location);
            mvptr->target_vehicle_id = object -> var.vehicleID;
            missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                    COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK,
                    COS_TERM, COS_LOSE);
        }
        else
        {
            mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
          if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                        mptr -> orientation[1]) < 0)
              printf ("missile_maverick_fly: TrackAcquire: %s\n",
                  TrackErrString ());
            missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                    SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
                    COS_LOSE);
        }
    }
```

APPENDIX I - miss_maverck.c

```
/*/
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_maverick_stop (mvptr);
        return;
    }
    else
    {
/*/
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
        {
            missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
            missile_maverick_stop (mvptr);
            return;
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_maverick_stop                               *
 * PARAMETERS:     mvptr - A pointer to the MAVERICK missile that  *
 *                     is to be stopped.                        *
 * RETURNS: none                                                *
 * PURPOSE: This routine causes all concerned to forget         *
 *          about the missile.  It should be called when        *
 *          the flyout of any MAVERICK missile is stopped       *
 *          (whether or not it has exploded).                   *
 *                                                              *
 *****************************************************************/

void missile_maverick_stop (mvptr)
MAVERICK_MISSILE *mvptr;
{
/*/
 *  If the world has been told to worry about this missile, tell it to stop
 *  then release missile memory for use by other missiles.
/*/
    if (mvptr->mptr.state == MAVERICK_FLYING)
        missile_util_comm_stop_missile (&(mvptr->mptr), MSL_TYPE_MISSILE);
    mvptr->mptr.state = MAVERICK_FREE;
    TrackSensorUnInit (mvptr -> sensor_id);
    mvptr -> sensor_id = NULL;
    mvptr -> object_being_tracked = NO_OBJECT; /* perhaps call break lock? */
```

```
}

static MAVERICK_MISSILE *missile_maverick_get_missile_from_sensor_id (sensor_id)
int sensor_id;
{
    register MAVERICK_MISSILE *mvptr = maverick_array;
    register int i;

    for (i = 0; i < num_mavericks; i++, mvptr++)
    {
      if (mvptr -> sensor_id == sensor_id)
          return (mvptr);
    }

    return (NULL);
}

static void missile_maverick_lock_handler (sensor_id, object)
int sensor_id;
TObjectP object;
{
    MAVERICK_MISSILE *mvptr;

    if (object == NO_OBJECT)
    {
        if (TrackDontLock (sensor_id, object) < 0)
          printf ("MaverickLockHandler: TrackDontLock: %s\n",
                  TrackErrString ());
      return;
    }

    if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
      != NULL)
    {
/* already tracking an object, but because of the delay from the TrackAqcuire
   call, the lock handler has been invoked again.  It does not matter if it is
   the same object or not as before.  Just do not lock again */

      if (mvptr -> object_being_tracked != NO_OBJECT)
      {
          if (TrackDontLock (sensor_id, object) < 0)
              printf ("MaverickLockHandler: TrackDontLock: %s\n",
                  TrackErrString ());
          return;
      }

      mvptr -> object_being_tracked = object;
      if (TrackLock (sensor_id, object) < 0)
          printf ("MaverickLockHandler: TrackLock: %s\n", TrackErrString ());
    }
    else
    {
        printf ("LockHandler: No missile for SensorId %d\n", sensor_id);
      if (TrackDontLock (sensor_id, object) < 0)
```

APPENDIX I - miss_maverck.c

```
      printf ("MaverickLockHandler: TrackDontLock: %s\n",
            TrackErrString ());
    }
}

static void missile_maverick_break_lock_handler (sensor_id, object)
int sensor_id;
TObjectP object;
{
    register MAVERICK_MISSILE *mvptr;
    if (object == NO_OBJECT)
        return;
    if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
      != NULL)
    {
      if (mvptr -> object_being_tracked == NO_OBJECT)
      {
            printf ("MaverickBreakLockHandler: BREAK LOCK BUT NOT LOCKED !!!\n");
          return;
      }

      if (mvptr -> object_being_tracked != object)
      {
          printf ("MaverickBreakLockHandler: BREAK LOCK ON UNKNOWN OBJECT!!!\n");
          return;
      }

      if (TrackBreakLock (sensor_id, object) < 0)
          printf ("MaverickBreakLockHandler: TrackBreakLock: %s\n",
                TrackErrString ());
      mvptr -> object_being_tracked = NO_OBJECT;
    }
    else
        printf ("BreakLockHandler: No missile for SensorId %d\n", sensor_id);
}

static REAL missile_maverick_detectibility (sensor_id, object, mav_loc,
                                  mav_boresight,
                                  flags)

int sensor_id;
TObjectP object;
VECTOR mav_loc;
VECTOR mav_boresight;
int flags;
{
    REAL detectibility;
    VECTOR target_location;
    VECTOR to_target;
    REAL dotProduct;
    MAVERICK_MISSILE *mvptr;

    /* Get location of object */

    GetLocationOfTObject (object, target_location);
```

APPENDIX I - miss_maverck.c

```
    /* Determine detectibility.  This is the cosine squared of the angle
     * between a vector from the sensor to the object and the boresight of
     * the sensor (for now).
     */

    /* Some of these computations may be duplicated in the tracking package.
     * May provide object calls to get them if that is more efficient.
     */

    vec_sub (target_location, mav_loc, to_target);
    dotProduct = vec_dot_prod (mav_boresight, to_target);
    detectibility = sign (dotProduct) * dotProduct * dotProduct /
                    vec_dot_prod (to_target, to_target);

    /* if the object is outside the detection cone of the sensor,
     * return a detectibility of 0.
     */

    if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
      != NULL)
    {
      switch (mvptr -> mptr.state)
      {
        case MAVERICK_READY:
          maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
          break;
        case MAVERICK_FLYING:
          maverick_cone_threshold = MAVERICK_HOLD_THRESHOLD;
          break;
        case MAVERICK_FREE:
        default:
          printf ("MaverickDetectibility: Maverick not READY or FLYING\n");
          maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
          break;
      }

      if (detectibility < maverick_cone_threshold)
            detectibility = 0.0;
    }
    else
    {
      printf ("MaverickDetectibility: no missile for sensorID %d\n",
            sensor_id);
    }

    return (detectibility);
}

static void  missile_maverick_object_update ()
{
}


/*
 * MissileMaverickSetPrelaunchIntervisibility
```

APPENDIX I - miss_maverck.c

```
 *
 * Called from command line switch processing code to set the intervisibility
 * interface to use and the way to init it.
 */
void missile_maverick_set_prelaunch_intervisibility_mode (mode)
char *mode;
{
    if (strlen (mode) > STRING_LEN)
    {
      printf ("missile_maverick_set_prelaunch__intervisibility: type string too
long\n");
      return;
    }
    strcpy (prelaunch_intervis_method, mode);
}

/*
 * MissileMaverickSetLaunchedIntervisibility
 *
 * Called from command line switch processing code to set the intervisibility
 * interface to use and the way to init it.
 */
void missile_maverick_set_launched_intervisibility_mode (mode)
char *mode;
{
    if (strlen (mode) > STRING_LEN)
    {
      printf ("missile_maverick_set_launched__intervisibility: type string too
long\n");
      return;
    }

    strcpy (in_flight_intervis_method, mode);
}

is_maverick_flying (sensor_id)
register int sensor_id;
{
    register int i;
    for (i = 0; i < num_mavericks; i++)
    {
      if (maverick_array[i].sensor_id == sensor_id)
      {
          if (maverick_array[i].mptr.state == MAVERICK_FLYING)
              return (TRUE);
          else
              return (FALSE);
      }
    }
    return (FALSE);
}

static void (*sensor_uninit_func) ();

void sensor_uninit_callback (sensor_id)
```

APPENDIX I - miss_maverck.c

```
int sensor_id;
{
    (*sensor_uninit_func) ();
}

missile_maverick_prepare_to_uninit_seeker (mvptr, uninit_func)
MAVERICK_MISSILE *mvptr;
void (*uninit_func) ();
{
    sensor_uninit_func = uninit_func;
    TrackSensorUnInitPrep (mvptr -> sensor_id, sensor_uninit_callback);
}
```

Appendix J - Source code listing for miss_nlos.c.

The following appendix contains the source code listing for miss_nlos.c for convenience in document maintenance and understanding of the CSU.

APPENDIX J - miss_nlos.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_nlo
s.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_nlos.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:13:50  cm-adst
 * R.Branson's changes for the weapson model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_nlos.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/*****************************************************************************
 *
 * Revisions:
 *
 *     Version        Date       Author       Title                      SP/CR Number
 *     _______        ________   _________    ______________________     ____________
 *
 *     1.2            10/23/92   R. Branson   Data File Initiali-
 *                                              zation
 *     1.3            10/30/92   R. Branson   Added pathname to data
 *                                              directory
 *     1.4            11/25/92   R. Branson   Changed %i to %d
 *
 *     1.5            01/19/93   P.Desmeules  Increased the size of the      31
 *                                              fgets to make sure the
 *                                              whole line is read in.
 *****************************************************************************/

/*****************************************************************************
 *
 *        SP/CR No.        Description of Modification
 *        __________       _____________________________________________________
 *
 *                         Hard coded defines changed to array elements.
 *                         Characteristics/parameter data array added.
 *                         Degree of polynomial data array added.
 *                         Added file reads for NLOS characteristics/
 *                            parameters, burn speed coefficients, and coast
 *                            speed coefficients.
 *
 *                         Added "/simnet/data/" to each data file pathname.
 *
 *****************************************************************************/

/*****************************************************************
 *                                                               *
 * FILE:       miss_nlos.c                                       *
 * AUTHOR:     Bryant Collard                                    *
```

APPENDIX J - miss_nlos.c

```
                                                                     *
* MAINTAINER: Bryant Collard                                         *
* PURPOSE:    This file contains routines which fly out a            *
*             missile with the characteristics of a NLOS             *
*             missile.                                               *
* HISTORY:    11/25/88 bryant: Creation                              *
*              4/24/89 bryant: Added static memory allocation        *
*             05/17/89 dan: changed hellfire to nlos                 *
*                                                                    *
*                                                                    *
* Copyright (c) 1988 BBN Systems and Technologies, Inc.              *
* All rights reserved.                                               *
*                                                                    *
*********************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "mass_stdc.h"
#include "dgi_stdg.h"
#include "sim_cig_if.h"
#include "protocol/pro_hdr.h"
#include "protocol/ammo.h"
#include "libmatrix.h"
#include "libmath.h"
#include "librva_util.h"
#include "libnear.h"

#include "miss_nlos.h"

#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)        a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/
#define NLOS_LOCK_THRESHOLD           nlos_miss_char[ 0]
#define NLOS_MAX_TURN_ANGLE           nlos_miss_char[ 1]
#define NLOS_VERTICAL_FLIGHT_TIME     nlos_miss_char[ 2]
#define NLOS_DECLINE_FLIGHT_TIME      nlos_miss_char[ 3]
                                             nlos_miss_char[ 4]
#define NLOS_LEVEL_FLIGHT_TIME
#define NLOS_ARM_TIME                 nlos_miss_char[ 5]
#define NLOS_BURNOUT_TIME             nlos_miss_char[ 6]
#define NLOS_MAX_FLIGHT_TIME          nlos_miss_char[ 7]
#define SPEED_0                       nlos_miss_char[ 8]
```

APPENDIX J - miss_nlos.c

```
#define SPEED_1                          nlos_miss_char[ 9]
/*#define THETA_0       0.046542113 */          /*0.013962634*/
#define THETA_0                          nlos_miss_char[10]

/*/
 *  Set parameters which will control flight trajectory behavior.
/*/

#define SIN_UNGUIDE                      nlos_miss_char[11]
#define COS_UNGUIDE                      nlos_miss_char[12]
#define SIN_CLIMB                   nlos_miss_char[13]
#define COS_CLIMB                   nlos_miss_char[14]
#define SIN_LOCK                    nlos_miss_char[15]
#define COS_LOCK                    nlos_miss_char[16]
#define COS_TERM                    nlos_miss_char[17]
#define COS_LOSE                    nlos_miss_char[18]

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/

#define NLOS_BURN_SPEED_DEG   nlos_miss_poly_deg[0]
#define NLOS_COAST_SPEED_DEG  nlos_miss_poly_deg[1]

/*/
 *  NLOS missile characteristic parameters initialized to default values.
/*/
static REAL nlos_miss_char[20] =
{
    0.953153895,  /* NLOS_LOCK_THRESHOLD  */
    0.03490659,        /* NLOS_MAX_TURN_ANGLE      radians/tick   */
   48.0,          /* NLOS_VERTICAL_FLIGHT_TIME */
  105.0,          /* NLOS_DECLINE_FLIGHT_TIME */
  140.0,          /* NLOS_LEVEL_FLIGHT_TIME */
   20.0,          /* NLOS_ARM_TIME           ticks (1.3 sec) */
   22.5,          /* NLOS_BURNOUT_TIME       ticks (1.5 sec) */
 8000.0,          /* NLOS_MAX_FLIGHT_TIME    ticks (120 sec) */
   11.33333333,        /* SPEED_0      */
    5.333333333,  /* SPEED_1      */
    /* THETA_0     0.046542113 */      /*0.013962634*/
    0.013962634,  /* THETA_0      */
    0.069756474,  /* SIN_UNGUIDE              4   deg */
    0.997564050,  /* COS_UNGUIDE              4   deg */
    0.004072424,  /* SIN_CLIMB                3.5 deg/sec */
    0.999991708,  /* COS_CLIMB                3.5 deg/sec */
    0.156434465,  /* SIN_LOCK                 9   deg */
    0.987688341,  /* COS_LOCK                 9   deg */
    0.984807753,  /* COS_TERM                 0   deg */
    0.939692621,  /* COS_LOSE                20   deg */
    0.0
};

/*/
```

APPENDIX J - miss_nlos.c

```
 *  The following terms set the order of the polynomials used to determine
 *  the speed and turn of the missile at any point in time.
/*/
static int nlos_miss_poly_deg[5] =
{
   1,       /* Speed before motor burnout. */
   3,       /* Speed after motor burnout. */
   0,
   0,
   0
};

/*/
 *  Coefficients for the speed polynomial before motor burnout.
/*/

static REAL nlos_burn_speed_coeff[5] =
{
    0.03333333,          /* a_0 - m/tick    ( 67.0 m/sec)              */
    1.25777777,          /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
    0.0,
    0.0,
    0.0
};

/*/
 *  Coefficients for the speed polynomial after motor burnout.
/*/

static REAL nlos_coast_speed_coeff[5] =
{
    30.46972849,         /* a_0 - m/tick    (327.2858074 m/sec)     */
    -9.7721160e-2,       /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
     1.2433925e-4,       /* a_2 - m/tick**3 (  0.8227650 m/sec**3) */
    -5.4061501e-8,       /* a_3 - m/tick**4 ( -0.0133200 m/sec**4) */
     0.0
};

static VECTOR nlos_initial_pos;
static VECTOR nlos_final_pos;
static VECTOR peak_target;
static VECTOR decline_target;
static VECTOR level_target;
static int nlos_target_id;
static int nlos_req_id;

/*/
 *  Declare static functions.
/*/

static void missile_nlos_stop ();


/*****************************************************************
 *                                                               *
```

APPENDIX J - miss_nlos.c

```
* ROUTINE: missile_nlos_init                                   *
* PARAMETERS:    mptr - a pointer to the NLOS to be                 *
*                 initialized.                                  *
* RETURNS: none                                                 *
* PURPOSE: This routine initializes the state of the            *
*          missile to indicate that it is available and         *
*          sets values that never change.                       *
*                                                               *
******************************************************************/

void missile_nlos_init (mptr)
MISSILE *mptr;
{
      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ NLOS missile file data $$$$\n"););

/*   DEFAULT CHARACTERISTICS DATA FOR miss_nlos.c READ FROM FILE               */
      fp = fopen("/simnet/data/ms_nl_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_nl_ch.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            nlos_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("nlos_miss_char(%3d) is%11.3f %s", i,
            nlos_miss_char[i], descript););
            ++i;
      }

      fclose(fp);
/*   END DEFAULT CHARACTERISTICS DATA FOR miss_nlos.c READ FROM FILE      */

/*   DEFAULT BURN SPEED DATA FOR miss_nlos.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_nl_bs.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_nl_bs.d\n");
            exit();
      }

      rewind(fp);

      /*    Read degree of polynomial */
```

APPENDIX J - miss_nlos.c

```
    fscanf(fp,"%d", &data_tmp_int);
    NLOS_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("nlos_miss_poly_deg(0) is%3d %s", NLOS_BURN_SPEED_DEG,
            descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          nlos_burn_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("nlos_burn_speed_coeff(%3d) is%11.3f %s", i,
                    nlos_burn_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_nlos.c READ FROM FILE     */

/*  DEFAULT COAST SPEED DATA FOR miss_nlos.c READ FROM FILE       */
    fp = fopen("/simnet/data/ms_nl_cs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_nl_cs.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    NLOS_COAST_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("nlos_miss_poly_deg(1) is%3d %s", NLOS_COAST_SPEED_DEG,
            descript););

    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          nlos_coast_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("nlos_coast_speed_coeff(%3d) is%11.3f %s", i,
                    nlos_coast_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_nlos.c READ FROM FILE    */

  mptr->state = FALSE;
  mptr->max_flight_time = NLOS_MAX_FLIGHT_TIME;
  mptr->max_turn_directions = 1;
  mptr->speed = SPEED_0;
```

APPENDIX J - miss_nlos.c

```
    mptr->cos_max_turn[0] = cos (NLOS_MAX_TURN_ANGLE);
    nlos_req_id = NEAR_NO_REQUEST_PENDING;
    nlos_target_id = vehicleIDIrrelevant;
}


/*****************************************************************
 *                                                               *
 * ROUTINE: missile_nlos_fire                                    *
 * PARAMETERS:    mptr - A pointer to the NLOS missile that *
 *                 is to be launched.                    *
 *          launch_point - The location in world             *
 *                         coordinates that the missile is  *
 *                         launched from.                *
 *          launch_to_world - The transformation matrix of  *
 *                            the launch platform to the    *
 *                            world.                     *
 *          launch_speed - The speed of the launch           *
 *                         platform (assumed to be in the   *
 *                         direction of the missile).  *
 *          tube - The tube the missile was launched from.  *
 * RETURNS: none                                                 *
 * PURPOSE: This routine performs the functions              *
 *          specifically related to the firing of a          *
 *          Hellfire missile.                                *
 *                                                               *
 *****************************************************************/

void missile_nlos_fire (mptr, launch_point, launch_to_world, launch_speed,
        tube)
MISSILE *mptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
{
/*/
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
/*/
    mptr->time = 0.0;
    mptr->speed = SPEED_0;
    vec_copy (launch_point, mptr->location);
    vec_copy (launch_point, nlos_initial_pos);
    mat_copy (launch_to_world, mptr->orientation);
    mptr->init_speed = launch_speed;
/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
                                          ammoHellfire, EFF_HELLFIRE,
                                          vehicleIDIrrelevant, targetUnknown,
                                          fuzePointDetonating, tube))
        return;
```

APPENDIX J - miss_nlos.c

```
/*/
 *  If all was successful, set the missile state to TRUE and return.
/*/
    mptr->state = TRUE;

    peak_target[X] = 0.0;
    peak_target[Y] = 1000.0;
    peak_target[Z] = 1000.0;
    vec_mat_mul (peak_target, mptr->orientation, peak_target);
    vec_add (mptr->location, peak_target, peak_target);
    printf("peak_target: x = %f, y = %f, z = %f\n",
      peak_target[X],
      peak_target[Y],
      peak_target[Z]);

    decline_target[X] = 0.0;
    decline_target[Y] = 1800.0;
    decline_target[Z] = 0.0;
    vec_mat_mul (decline_target, mptr->orientation, decline_target);
    vec_add (mptr->location, decline_target, decline_target);
    printf("decline_target: x = %f, y = %f, z = %f\n",
      decline_target[X],
      decline_target[Y],
      decline_target[Z]);

    level_target[X] = 0.0;
    level_target[Y] = 2000.0;
    level_target[Z] = 300.0;
    vec_mat_mul (level_target, mptr->orientation, level_target);
    vec_add (mptr->location, level_target, level_target);
    printf("level_target: x = %f, y = %f, z = %f\n",
      level_target[X],
      level_target[Y],
      level_target[Z]);

    return;
}

/*****************************************************************
 *                                                       *
 * ROUTINE: missile_nlos_fly                             *
 * PARAMETERS:    mptr - A pointer to the NLOS missile that *
 *                  is to be flown out.                  *
 *          target_location - The location in world        *
 *                           coordinates of the target.    *
 * RETURNS: none                                         *
 * PURPOSE: This routine performs the functions           *
 *          specifically related to the flying a NLOS *
 *          missile.                                     *
 *                                                       *
 *****************************************************************/

void missile_nlos_fly (mptr, nlos_target_loc, target_scheme)
MISSILE *mptr;
VECTOR nlos_target_loc;
```

APPENDIX J - miss_nlos.c

```
int target_scheme;
{
   register REAL time;          /* The current time after launch (ticks). */
   register REAL temp;
   VehicleAppearancePDU *target;    /* A pointer to the target vehicles
                                 appearance packet. */


/*
   timed_printf("target_scheme = %d\nloc %f %f %f\n",
     target_scheme,
     nlos_target_loc[0],
     nlos_target_loc[1],
     nlos_target_loc[2]
     );
*/
/*/
 *  Set and _time_.  This is created mostly for increased readablity.
/*/
   time = mptr->time;

   if (time > 800.0)
     mptr->speed = SPEED_1;

/*/
 *  choose the correct targetttting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
   printf("extra_waypoint: %f %f %f\n",
     mptr->location[0],
     mptr->location[1],
     mptr->location[2]);

   if (time < NLOS_VERTICAL_FLIGHT_TIME)
     missile_nlos_fly_to_point(mptr, peak_target);
   else if (time < NLOS_DECLINE_FLIGHT_TIME)
     missile_nlos_fly_to_point(mptr, decline_target);
   else if (time < NLOS_LEVEL_FLIGHT_TIME)
     {
     level_target[Z] = mptr->location[Z];
     missile_nlos_fly_to_point(mptr, level_target);
     }
   else
     {
     switch (target_scheme)
          {
          case NLOS_FLY_TO_POINT_IN_SPACE:
            missile_nlos_fly_to_point(mptr, nlos_target_loc);
            break;

          case NLOS_FLY_TO_POINT_RELATIVE:
            missile_target_nlos(mptr, nlos_target_loc);
            break;

          case NLOS_FLY_TO_TARGET:
            target = near_get_preferred_veh_near_vector (
```

```
                              &nlos_target_id,
                                    RVA_ALL_VEH,
                              mptr->location,
                              mptr->orientation[1],
                                    NLOS_LOCK_THRESHOLD,
                              &nlos_req_id);
            if (target != NULL)
                {
                timed_printf("miss_nlos: target locked on\n");
                missile_target_pursuit (mptr, target);
                }
            else
                {
                missile_target_unguided(mptr);
                }
            break;

          default:
            printf("missile_nlos_fly: bad target_scheme\n");
            break;
          }
      }

/*/
 *  check to see if the missile is "out of gas"
/*/
    if (mptr->time > 1500.0)
      mptr->target[Z] = 0.0;

/*/
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_nlos_stop (mptr);
      if (target_scheme == NLOS_FLY_TO_TARGET)
          {
          nlos_target_id = vehicleIDIrrelevant;
          nlos_req_id = NEAR_NO_REQUEST_PENDING;
          }
        return;
    }
    else
    {
/*/
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
        {
            missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
            missile_nlos_stop (mptr);
            return;
```

APPENDIX J - miss_nlos.c

```
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************
 *                                                              *
 * ROUTINE: missile_nlos_stop                                   *
 * PARAMETERS:    mptr - A pointer to the NLOS missile that *
 *                  is to be stopped.                         *
 * RETURNS: none                                                *
 * PURPOSE: This routine causes all concerned to forget        *
 *          about the missile.  It should be called when       *
 *          the flyout of any NLOS missile is stopped *
 *          (whether or not it has exploded).  Note that       *
 *          this routine can only be called within this        *
 *          module.                                          *
 *                                                              *
 *****************************************************************/

static void missile_nlos_stop (mptr)
MISSILE *mptr;
{
/*/
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
/*/
    printf("initial_pos = %f %f %f\n",
            nlos_initial_pos[0],
            nlos_initial_pos[1],
            nlos_initial_pos[2]);

    printf("final_position = %f %f %f\n",
            mptr->location[0],
            mptr->location[1],
            mptr->location[2]);

    missile_util_comm_stop_missile (mptr, MSL_TYPE_MISSILE);
    mptr->state = FALSE;
}
```

Appendix K - Source code listing for miss_stinger.c.

The following appendix contains the source code listing for miss_stinger.c for convenience in document maintenance and understanding of the CSU.

APPENDIX K - miss_stinger.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_sti
nger.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_stinger.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_stinger.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/*************************************************************************
*
* Revisions:
*
*     Version        Date       Author       Title                    SP/CR Number
*     _______        ________   __________   ____________________     ____________
*
*     1.2            10/23/92   R. Branson   Data File Initiali-
*                                              zation
*     1.3            10/30/92   R. Branson   Added pathname to data
*                                              directory
*     1.4            11/25/92   R. Branson   Changed %i to %d
*
*     1.5            01/12/93   P.J.Desmeules Increased the size         31
*                                              of the fgets to make
*                                              sure the whole line is
*                                              read.
*
*************************************************************************/

/*************************************************************************
*
*         SP/CR No.         Description of Modification
*         __________        ____________________________________________
*
*
*                           Hard coded defines changed to array elements.
*                           Characteristics/parameter data array added.
*                           Degree 0f polynomial data array added.
*                           Added file reads for stinger characteristics/
*                              parameters, burn speed coefficients, and coast
*                              speed coefficients.
*
*                           Added "/simnet/data/" to each data file pathname.
*
*************************************************************************/

/************************************************************
 *                                                          *
 * FILE:       miss_stinger.c                               *
 * AUTHOR:     Bryant Collard                               *
 * MAINTAINER: Bryant Collard                               *
```

APPENDIX K - miss_stinger.c

```
* PURPOSE:    This file contains routines which fly out a        *
*             missile with the characteristics of a STINGER      *
*             missile.                                           *
* HISTORY:    12/08/88 bryant: Creation                          *
*             04/24/89 bryant: Added static memory allocation    *
*             08/07/90 bryant: NIU librva modifications.         *
*                                                                *
*                                                                *
* Copyright (c) 1988 BBN Systems and Technologies, Inc.          *
* All rights reserved.                                           *
*                                                                *
******************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libnear.h"
/*-- need Range_Squared info --*/
#include "libhull.h"
#include "libkin.h"
/*----------------------------*/
#include "miss_stinger.h"

#include "libmissile.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)       a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define STINGER_BURNOUT_TIME      stinger_miss_char[ 0]
#define STINGER_MAX_FLIGHT_TIME   stinger_miss_char[ 1]
#define STINGER_LOCK_THRESHOLD    stinger_miss_char[ 2]
#define SPEED_0                   stinger_miss_char[ 3]
#define THETA_0                   stinger_miss_char[ 4]
#define INVEST_DIST_SQ            stinger_miss_char[ 5]
#define FUZE_DIST_SQ              stinger_miss_char[ 6]

/*/
```

APPENDIX K - miss_stinger.c

```
 *  Define the states the _STINGER_MISSILE_ can be in.
/*/

#define STINGER_FREE    0      /* No missile assigned. */
#define STINGER_READY   1      /* Missile assigned to ready state. */
#define STINGER_FLYING  2      /* Missile assigned to flying state. */

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed of the missile at any point in time.
/*/
static int stinger_miss_poly_deg[2] =
{
    1,      /* burn speed poly degree  */
    3   /* coast speed poly degree */
};

/*/
 *  Stinger missile characteristic parameters initialized to default values.
/*/
static REAL stinger_miss_char[15] =
{
    19.125,       /* ticks (1.275 sec) */
   400.000,       /* ticks (26.667 sec) */
     0.953153895,       /* cos (12.5 deg) ** 2 */
    53.33333333,        /* m/tick (800 m/sec) */
     0.0174,            /* rad/tick (15.0 deg/sec) */
 90000.0,               /* (300 m) ** 2 */
   400.0,               /* (20 m) ** 2 */
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
};

/*/
 *  Coefficients for the speed polynomial before motor burnout initialized to
 *  default values.
/*/

static REAL stinger_burn_speed_coeff[STINGER_BURN_SPEED_DEG + 1] =
{
    1.9,                 /* a_0 - m/tick */
    2.689324619          /* a_1 - m/tick**2 */
};

/*/
 *  Coefficients for the speed polynomial after motor burnout initialized to
 *  default values.
/*/
```

APPENDIX K - miss_stinger.c

```
static REAL stinger_coast_speed_coeff[STINGER_COAST_SPEED_DEG + 1] =
{
    56.73662833,          /* a_0 - m/tick */
    -0.182369351,         /* a_1 - m/tick**2 */
    2.3302001e-4,         /* a_2 - m/tick**3 */
    -1.0176282e-7         /* a_3 - m/tick**4 */
};

/*/
 *  Memory for the missiles is declared in vehicle specific code.  During
 *  initialization, a pointer is assigned to this memory then all memory
 *  issues are dealt with in this module.
/*/

static STINGER_MISSILE *stinger_array;  /* A pointer to missile memory. */
static int num_stingers;                /* The number of defined missiles. */

static ObjectType stinger_ammo_type = munition_US_Stinger;
static REAL
    max_range_limit,     /* [ MISSILE_US_MAX_RANGE_LIMIT ]                    */
    max_range_squared,   /* [ MISSILE_US_MAX_RANGE_LIMIT ^ 2 ]                */
    speed_factor;        /* [ MISSILE_US_SPEED_FACTOR ]                       */

/*/
 *  Declare static functions.
/*/

static void missile_stinger_fly ();


/******************************************************************
 *                                                                *
 * ROUTINE: missile_stinger_init                                  *
 * PARAMETERS:     missile_array - A pointer to an array of       *
 *                           STINGER missiles defined in          *
 *                           vehicle specific code.               *
 *          num_missiles - The number missiles defined in         *
 *                          _missile_array_.                      *
 * RETURNS: none                                                  *
 * PURPOSE: This routine copies the parameters into               *
 *          variables static to this module and initializes       *
 *          the state of all the missiles.  It also               *
 *          initializes the proximity fuze.                       *
 *                                                                *
 ******************************************************************/

void missile_stinger_init (missile_array, num_missiles)
STINGER_MISSILE missile_array[];
int num_missiles;
{
    int i;    /* A counter. */
            int   j;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
```

APPENDIX K - miss_stinger.c

```
    FILE  *fp;

    P(printf("$$$$$ STINGER missile file data $$$$\n"););

/*  DEFAULT CHARACTERISTIC DATA FOR miss_stinger.c READ FROM FILE */
    fp = fopen("/simnet/data/ms_st_ch.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_st_ch.d\n");
          exit();
    }

    rewind(fp);

    /*    Read array data    */
    j=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          stinger_miss_char[j] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("stinger_miss_char(%3d) is%11.3f %s", j,
                stinger_miss_char[j],
                      descript););
          ++j;
    }

    fclose(fp);
/*  END DEFAULT CHARACTERISTIC DATA FOR miss_stinger.c READ FROM FILE   */

/*  DEFAULT BURN SPEED DATA FOR miss_stinger.c READ FROM FILE     */
    fp = fopen("/simnet/data/ms_st_bs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_st_bs.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial     */

    fscanf(fp,"%d", &data_tmp_int);
    stinger_miss_poly_deg[0] = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("stinger_miss_poly_deg(0) is%3d %s",
    stinger_miss_poly_deg[0], descript););

    /*    Read array data    */
    j=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          stinger_burn_speed_coeff[j] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("stinger_burn_speed_coeff(%3d) is%11.3f %s", j,
                stinger_burn_speed_coeff[j],
                      descript););
          ++j;
```

```
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_stinger.c READ FROM FILE */

/*  DEFAULT COAST SPEED DATA FOR miss_stinger.c READ FROM FILE    */
    fp = fopen("/simnet/data/ms_st_cs.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_st_cs.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial     */

    fscanf(fp,"%d", &data_tmp_int);
    stinger_miss_poly_deg[1] = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("stinger_miss_poly_deg(1) is%3d %s",
          stinger_miss_poly_deg[1], descript););

    /*    Read array data   */
    j=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          stinger_coast_speed_coeff[j] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("stinger_coast_speed_coeff(%3d) is%11.3f %s", j,
                stinger_coast_speed_coeff[j],
                      descript););
          ++j;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_stinger.c READ FROM FILE      */

   num_stingers = num_missiles;
   stinger_array = missile_array;
   for (i = 0; i < num_missiles; i++)
   {
      stinger_array[i].mptr.state = STINGER_FREE;
      stinger_array[i].mptr.max_flight_time = STINGER_MAX_FLIGHT_TIME;
      stinger_array[i].mptr.max_turn_directions = 1;
   }
   speed_factor = MISSILE_US_SPEED_FACTOR;
   max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
   max_range_squared = max_range_limit * max_range_limit;
   stinger_ammo_type = munition_US_Stinger;
/*/
 *  Initialize the proximity fuze.
/*/
   missile_fuze_prox_init ();
}
```

APPENDIX K - miss_stinger.c

```
void missile_stinger_set_speed_factor( scale_speed )
REAL scale_speed;
{
    speed_factor = scale_speed;
}

void missile_stinger_set_max_range_limit( limit_range )
REAL limit_range;
{
    max_range_limit = limit_range;
    max_range_squared = max_range_limit * max_range_limit;
}

void missile_stinger_set_ammo_type( ammo )
ObjectType ammo;
{
    stinger_ammo_type = ammo;
}


/****************************************************************
 *                                                              *
 * ROUTINE: missile_stinger_ready                               *
 * PARAMETERS:    none                                          *
 * RETURNS: A pointer to a missile that is currently            *
 *          available.                                          *
 * PURPOSE: This routine finds, if possible, a missile that     *
 *          is not being used, puts it in a ready state and     *
 *          returns a pointer to it.                            *
 *                                                              *
 ****************************************************************/

STINGER_MISSILE *missile_stinger_ready ()
{
    int i;     /* A counter. */
/**/
 *  Try to find a free missile.
/**/
    for (i = 0; i < num_stingers; i++)
    {
/**/
 *      If a free missile is found, put it in a ready state, clear the target
 *      ID and return a pointer to it.
/**/
        if (stinger_array[i].mptr.state == STINGER_FREE)
        {
            stinger_array[i].mptr.state = STINGER_READY;
            stinger_array[i].target_vehicle_id.vehicle = vehicleIrrelevant;
            return (&stinger_array[i]);
        }
    }
/**/
 *  If no free missile is found, return a NULL pointer.
/**/
```

```
    return (NULL);
}


/******************************************************************
 *                                                        *
 * ROUTINE: missile_stinger_pre_launch                         *
 * PARAMETERS:    sptr - A pointer to the missile that is to be     *
 *                 serviced.                              *
 *          launch_point - The location of the missile in    *
 *                         world coordinates.             *
 *          launch_to_world - The transformation matrix of  *
 *                            the missile to the world.    *
 *              veh_list - Vehicle list ID.                      *
 * RETURNS: none                                          *
 * PURPOSE: This routine is called after a missile has been *
 *          readied and before it has been launched.  It     *
 *          determines if the seeker head can see a target  *
 *          and, if it can see a target, stores its         *
 *          position.                                     *
 *                                                        *
 ******************************************************************/

void missile_stinger_pre_launch (sptr, launch_point, launch_to_world, veh_list)
STINGER_MISSILE *sptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
int veh_list;
{
    VehicleAppearanceVariant *target;  /* A pointer to the target vehicles
                                          appearance packet. */
/*/
 *  Try to find a target.
/*/
    target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
            veh_list, launch_point, launch_to_world[1],
            STINGER_LOCK_THRESHOLD);
/*/
 *  If a target is found, store its location.
/*/
    if (target != NULL)
    {
        sptr->target_vehicle_id = target->vehicleID;
        missile_target_pursuit (&(sptr->mptr), target->location);
    }
    else
        sptr->target_vehicle_id.vehicle = vehicleIrrelevant;
}


/******************************************************************
 *                                                        *
 * ROUTINE: missile_stinger_fire                               *
 * PARAMETERS:    sptr - A pointer to the STINGER missile that     *
 *                 is to be launched.                          *
```

```
*           launch_point - The location in world                *
*                          coordinates that the missile is      *
*                          launched from.                       *
*           launch_to_world - The transformation matrix of      *
*                             the launch platform to the        *
*                             world.                            *
*           launch_speed - The speed of the launch              *
*                          platform (assumed to be in the       *
*                          direction of the missile).           *
*           tube - The tube the missile was launched from.      *
* RETURNS:  TRUE for a successful launch and FALSE for an       *
*           unsuccessful launch.
* PURPOSE:  This routine performs the functions                 *
*           specifically related to the firing of a             *
*           STINGER missile.                                    *
*                                                               *
****************************************************************/

int missile_stinger_fire (sptr, launch_point, launch_to_world, launch_speed,
        tube)
STINGER_MISSILE *sptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
{
    int i;                      /* Counter. */
    MISSILE *mptr;              /* Pointer to the particular generic missile
                                   pointed at by _sptr_. */
/*/
 *  Get a pointer to the generic elements of the STINGER missile.  This
 *  improves code readability.
/*/
    mptr = &(sptr->mptr);
/*/
 *  Set the initial time, location, orientation and speed of the generic
 *  missile.
/*/
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (launch_to_world, mptr->orientation);
    mptr->speed = launch_speed +
        (speed_factor *
         missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
                                 stinger_burn_speed_coeff, 0.0));
    mptr->init_speed = launch_speed;
/*/
 *  Indicate that the proximity fuze has no vehicles it is tracking.
/*/
    sptr->pptr = NULL;
/*/
 *  Determine range equations for intercept targeting.
/*/
    sptr->stinger_burn_range_coeff[0] = 0.0;
    for (i = 1; i <= STINGER_BURN_SPEED_DEG + 1; i++);
```

```
    {
        sptr->stinger_burn_range_coeff[i] = (1.0 / ((REAL) i)) *
                stinger_burn_speed_coeff[i - 1];
    }
    sptr->stinger_burn_range_coeff[1] += launch_speed;
    missile_target_intercept_find_poly (STINGER_COAST_SPEED_DEG, launch_speed,
            stinger_coast_speed_coeff, sptr->stinger_coast_range_coeff,
            sptr->stinger_coast_range_2_coeff);
/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, release the missile memory and return FALSE.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (stinger_ammo_type),
            stinger_ammo_type, stinger_ammo_type,
            &(sptr->target_vehicle_id), targetIsVehicle, objectIrrelevant,
            tube))
    {
        mptr->state = STINGER_FREE;
        return (FALSE);
    }
/*/
 *  If all was successful, set the missile state to STINGER_FLYING and
 *  return TRUE.
/*/
    mptr->state = STINGER_FLYING;
    return (TRUE);
}


/*****************************************************************
 *                                                          *
 * ROUTINE: missile_stinger_fly_missiles                         *
 * PARAMETERS:    veh_list - Vehicle list ID.                          *
 * RETURNS: none                                             *
 * PURPOSE: This routine flies out all missiles in a         *
 *          flying state.                                        *
 *                                                          *
 *****************************************************************/

void missile_stinger_fly_missiles (veh_list)
int veh_list;
{
    int i;      /* A counter. */
/*/
 *  Fly out all flying missiles.
/*/
    for (i = 0; i < num_stingers; i++)
    {
        if (stinger_array[i].mptr.state == STINGER_FLYING)
            missile_stinger_fly (&(stinger_array[i]), veh_list);
    }
}
```

APPENDIX K - miss_stinger.c

```
/******************************************************************
 *                                                                *
 * ROUTINE: missile_stinger_fly                                   *
 * PARAMETERS:    sptr - A pointer to the STINGER missile that    *
 *                 is to be flown out.                            *
 *              veh_list - Vehicle list ID.                       *
 * RETURNS: none                                                  *
 * PURPOSE: This routine performs the functions                   *
 *          specifically related to the flying a STINGER          *
 *          missile.                                              *
 *                                                                *
 ******************************************************************/

static void missile_stinger_fly (sptr, veh_list)
STINGER_MISSILE *sptr;
int veh_list;
{
    register MISSILE *mptr;         /* A pointer to the generic aspects of
                                       _sptr_. */
    REAL time;                      /* The current time after launch (ticks). */
    VehicleAppearanceVariant
 ,      *target;                    /* A pointer to the targets appearance
                                       packet. */

/**/
 *  Set _mptr_ and _time_.  These values are created mostly for increased
 *  readablity.
/**/
    mptr = &(sptr->mptr);
    time = mptr->time;
/**/
 *  Find the current missile speed and the cosine of the maximum allowed turn
 *  angle.  The equations used are different before and after motor burnout.
/**/
    if (time < STINGER_BURNOUT_TIME)
    {
        mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
                stinger_burn_speed_coeff, time) + mptr->init_speed;
    }
    else
    {
        mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
                stinger_coast_speed_coeff, time) + mptr->init_speed;
    }
/**/
 *  Note that this is a temporary method of finding turn angle.
/**/
    mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
            mptr->init_speed)) * THETA_0);
/**/
 *  Try to find a target.  If one is found, fly towards it in the
 *  proper trajectory, otherwise, fly in a straight line.
/**/
    target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
            veh_list, mptr->location, mptr->orientation[1],
            STINGER_LOCK_THRESHOLD);
```

APPENDIX K - miss_stinger.c

```
    if( max_range_limit > 0  &&
       kinematics_range_squared (veh_kinematics, mptr->location) >
       max_range_squared )
        missile_target_ground( mptr );
    else if (target != NULL)
    {
        sptr->target_vehicle_id = target->vehicleID;
        if (time < STINGER_BURNOUT_TIME)
            missile_target_intercept_pre_burnout (mptr, target,
                    sptr->stinger_burn_range_coeff, STINGER_BURNOUT_TIME,
                    STINGER_BURN_SPEED_DEG + 1,
                    sptr->stinger_coast_range_coeff,
                    sptr->stinger_coast_range_2_coeff,
                    STINGER_COAST_SPEED_DEG + 1);

        else
            missile_target_intercept (mptr, target,
                    sptr->stinger_coast_range_coeff,
                    sptr->stinger_coast_range_2_coeff,
                    STINGER_COAST_SPEED_DEG + 1);
    }
    else
    {
        sptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        missile_target_unguided (mptr);
    }
/*/
 *  Try to actually fly the missile.  If this fails, stop the missile
 *  altogether and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_stinger_stop (sptr);
        return;
    }
    else
    {
/*/
 *      If the missile successfully flew, process the proximity fuze.
/*/
        if (sptr->target_vehicle_id.vehicle == vehicleIrrelevant)
            missile_fuze_prox (mptr, MSL_TYPE_MISSILE, PROX_FUZE_ON_ALL_VEH,
                    &(sptr->target_vehicle_id), &(sptr->pptr),
                    veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
        else
            missile_fuze_prox (mptr, MSL_TYPE_MISSILE, PROX_FUZE_ON_ONE_VEH,
                    &(sptr->target_vehicle_id), &(sptr->pptr),
                    veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);

/*/
 *      If the missile has intersected of self detonated, blow it up, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE))
        {
            missile_stinger_stop (sptr);
            return;
```

APPENDIX K - miss_stinger.c

```
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/****************************************************************
 *                                                             *
 * ROUTINE: missile_stinger_stop                               *
 * PARAMETERS:    sptr - A pointer to the STINGER missile that *
 *                 is to be stopped.                           *
 * RETURNS: none                                               *
 * PURPOSE: This routine causes all concerned to forget        *
 *          about the missile.  It should be called when       *
 *          the flyout of any STINGER missile is stopped       *
 *          (whether or not it has exploded).                  *
 *                                                             *
 ****************************************************************/

void missile_stinger_stop (sptr)
STINGER_MISSILE *sptr;
{
/*/
 *  If the missile has been fired, tell the world to stop it and clear the
 *  proximity fuze targets.  Release missile memory for use by other missiles.
/*/
    if (sptr->mptr.state == STINGER_FLYING)
    {
        missile_util_comm_stop_missile (&(sptr->mptr), MSL_TYPE_MISSILE);
        missile_fuze_prox_stop (&(sptr->pptr));
    }
    sptr->mptr.state = STINGER_FREE;
}
```

Appendix L - Source code listing for miss_tow.c.

The following appendix contains the source code listing for miss_tow.c for convenience in document maintenance and understanding of the CSU.

APPENDIX L - miss_tow.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_tow
.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_tow.c,v $
 * Revision 1.4  1993/01/28  23:22:08  cm-adst
 * P.DesMeules changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:14:12  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_tow.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";

/***************************************************************************
*
* Revisions:
*
*     Version       Date       Author        Title                     SP/CR Number
*     _______       ________   ________      ___________________       __________
*
*     1.2           10/23/92   R. Branson    Data File Initiali-
*                                              zation
*     1.3           10/30/92   R. Branson    Added pathname to data
*                                              directory
*     1.4           11/25/92   R. Branson    Changed %i to %d
*
*     1.5           01/19/93   P.Desmeules   Increased the size of the     31
*                                              fgets to make sure the
*                                              whole line is read in.
***************************************************************************/

/***************************************************************************
*
*        SP/CR No.          Description of Modification
*        __________         ____________________________________________
*
*                           Hard coded defines changed to array elements.
*                           Characteristics/parameter data array added.
*                           Degree of polynomial data array added.
*                           Added file reads for TOW characteristics/parameters,
*                              burn speed coefficients, coast speed coefficients,
*                              burn turn coefficients, and coast turn coeffi-
*                              coefficients.
*
*                           Added "/simnet/data/" to each data file pathname.
*
***************************************************************************/

/*************************************************************
 *                                                           *
 * FILE:        miss_tow.c                                   *
```

APPENDIX L - miss_tow.c

```
* AUTHOR:      Bryant Collard                                        *
* MAINTAINER: Bryant Collard                                         *
* PURPOSE:     This file contains routines which fly out a           *
*              missile with the characteristics of a TOW             *
*              missile.                                              *
* HISTORY:     10/31/88 bryant: Creation                             *
*               4/26/89 bryant: Added statically allocated mem       *
*                                                                    *
*                                                                    *
*                                                                    *
* Copyright (c) 1988 BBN Systems and Technologies, Inc.              *
* All rights reserved.                                               *
*                                                                    *
*********************************************************************/

#include "stdio.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmatrix.h"
#include "libmap.h"
/*-- need Range_Squared info --*/
#include "libhull.h"
#include "libkin.h"
/*----------------------------*/
#include "miss_tow.h"

#include "libmissile.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)        a
#else
#define P(a)
#endif

/*/
 *  Define missile characteristics.
/*/

#define TOW_BURNOUT_TIME       tow_miss_char[0]
#define TOW_RANGE_LIMIT_TIME   tow_miss_char[1]
#define TOW_MAX_FLIGHT_TIME    tow_miss_char[2]

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed or cosine of the maximum allowed turn rate of the missile
 *  at any point in time.
/*/
```

APPENDIX L - miss_tow.c

```
#define TOW_BURN_SPEED_DEG  tow_miss_poly_deg[0]
#define TOW_COAST_SPEED_DEG tow_miss_poly_deg[1]
#define TOW_BURN_TURN_DEG   tow_miss_poly_deg[2]
#define TOW_COAST_TURN_DEG  tow_miss_poly_deg[3]

/*/
 *  Tow missile characteristic parameters initialized to default values.
/*/
static REAL tow_miss_char[5] =
{
   24.0,     /* ticks (1.6 sec) */
  268.35,    /* ticks (17.89 sec) */
  300.00,    /* ticks - cos of max turn > 1.0 beyond this point */
    0.0,
    0.0
};

/*/
 *  The following terms set the order of the polynomials used to determine
 *  the speed and turn of the missile at any point in time.
/*/
static int tow_miss_poly_deg[5] =
{
   2,        /* Speed before motor burnout. */
   3,        /* Speed after motor burnout. */
   1,        /* Cosine of max turn before burnout. */
   3,        /* Cosine of max turn after burnout. */
   0         /* not used.  */
};

/*/
 *  Coefficients for the speed polynomial before motor burnout initialized
 *  to default values.
/*/

static REAL tow_burn_speed_coeff[5] =
{
    4.466666667,         /* a_0 - m/tick (67.0 m/sec) */
    1.222103405,         /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
   -0.024532086,         /* a_2 - m/tick**3 (-82.7057910 m/sec**3) */
    0.0,
    0.0
};

/*/
 *  Coefficients for the speed polynomial after motor burnout.
/*/

static REAL tow_coast_speed_coeff[5] =
{
    21.81905383,         /* a_0 - m/tick (327.2858074 m/sec) */
    -9.5382019e-2,       /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
     2.4378222e-4,       /* a_2 - m/tick**3 (0.8227650 m/sec**3) */
    -2.6311111e-7,       /* a_3 - m/tick**4 (-0.0133200 m/sec**4) */
     0.0
```

```
};

/*/
 *  Coefficients for the cosine of max turn polynomials before motor burnout.
 *  The structure _MAX_COS_COEFF_ is used to store the values for the turn
 *  sideways, up, and down polynomials along with their order.
/*/

static MAX_COS_COEFF tow_burn_turn_coeff =
{
    1,                      /* Order of the polynomials. */
    {
                            /* Sidewards turn. */
        0.999976868652,     /* a_0 - cos(rad)/tick */
       -3.5933955e-7        /* a_1 - cos(rad)/tick**2 */
    },
    {
                            /* Upwards turn. */
        0.999960667258,     /* a_0 - cos(rad)/tick */
       -3.1492328e-6        /* a_1 - cos(rad)/tick**2 */
    },
    {
                            /* Downwards turn. */
        0.999978909989,     /* a_0 - cos(rad)/tick */
       -7.8194991e-9        /* a_1 - cos(rad)/tick**2 */
    }
};

/*/
 *  Coefficients for the cosine of max turn polynomials after motor burnout.
/*/

static MAX_COS_COEFF tow_coast_turn_coeff =
{
    3,                      /* Order of the polynomials. */
    {
                            /* Sidewards turn. */
        0.99995112518,      /* a_0 - cos(rad)/tick */
        8.96333e-7,         /* a_1 - cos(rad)/tick**2 */
       -5.995375e-9,        /* a_2 - cos(rad)/tick**3 */
        1.162225e-11        /* a_3 - cos(rad)/tick**4 */
    },
    {
                            /* Upwards turn. */
        0.9998498495,       /* a_0 - cos(rad)/tick */
        1.657779e-6,        /* a_1 - cos(rad)/tick**2 */
       -8.231861e-9,        /* a_2 - cos(rad)/tick**3 */
        1.381832e-11        /* a_3 - cos(rad)/tick**4 */
    },
    {
                            /* Downwards turn. */
        0.9999714014,       /* a_0 - cos(rad)/tick */
        3.382077e-7,        /* a_1 - cos(rad)/tick**2 */
       -1.601259e-9,        /* a_2 - cos(rad)/tick**3 */
        2.623014e-12        /* a_3 - cos(rad)/tick**4 */
```

APPENDIX L - miss_tow.c

```
    }
};

static ObjectType tow_ammo_type = munition_US_TOW;
static REAL
    max_range_limit,     /* [ MISSILE_US_MAX_RANGE_LIMIT ]                    */
    max_range_squared,   /* [ MISSILE_US_MAX_RANGE_LIMIT ^ 2 ]                */
    speed_factor;        /* [ MISSILE_US_SPEED_FACTOR ]                       */

/*/
 *  Declare static functions.
/*/
static void missile_tow_stop ();


/*****************************************************************
 *                                                               *
 * ROUTINE: missile_tow_init                                     *
 * PARAMETERS:    tptr - a pointer to the TOW to be              *
 *                 initialized.                                  *
 * RETURNS: none                                                 *
 * PURPOSE: This routine initializes the state of the            *
 *          missile to indicate that it is available and         *
 *          sets values that never change.                       *
 *                                                               *
 *****************************************************************/

void missile_tow_init (tptr)
TOW_MISSILE *tptr;
{
      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ TOW missile file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR miss_tow.c READ FROM FILE              */
      fp = fopen("/simnet/data/ms_tw_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_tw_ch.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            tow_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_miss_char(%3d) is%11.3f %s", i, tow_miss_char[i],
                        descript););
```

APPENDIX L - miss_tow.c

```
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR miss_tow.c READ FROM FILE        */

/*  DEFAULT BURN SPEED DATA FOR miss_tow.c READ FROM FILE          */
      fp = fopen("/simnet/data/ms_tw_bs.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_tw_bs.d\n");
            exit();
      }

      rewind(fp);

      /*    Read degree of polynomial */

      fscanf(fp,"%d", &data_tmp_int);
      TOW_BURN_SPEED_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("tow_miss_poly_deg(0) is%3d %s", TOW_BURN_SPEED_DEG,
                  descript););

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            tow_burn_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_burn_speed_coeff(%3d) is%11.3f %s", i,
                        tow_burn_speed_coeff[i], descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR miss_tow.c READ FROM FILE      */

/*  DEFAULT COAST SPEED DATA FOR miss_tow.c READ FROM FILE         */
      fp = fopen("/simnet/data/ms_tw_cs.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_tw_cs.d\n");
            exit();
      }

      rewind(fp);

      /*    Read degree of polynomial */

      fscanf(fp,"%d", &data_tmp_int);
      TOW_COAST_SPEED_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("tow_miss_poly_deg(1) is%3d %s", TOW_COAST_SPEED_DEG,
                  descript););

      /*    Read array data    */
```

```
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          tow_coast_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_coast_speed_coeff(%3d) is%11.3f %s", i,
                            tow_coast_speed_coeff[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT COAST SPEED DATA FOR miss_tow.c READ FROM FILE     */

/*  DEFAULT BURN TURN DATA FOR miss_tow.c READ FROM FILE           */
    fp = fopen("/simnet/data/ms_tw_bt.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_tw_bt.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    TOW_BURN_TURN_DEG = data_tmp_int;
    tow_burn_turn_coeff.deg = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("tow_miss_poly_deg(2) is%3d %s", TOW_BURN_TURN_DEG,
               descript););

    /*    Read array data    */

    for (i=0; i <= data_tmp_int; i++) {
          fscanf(fp,"%f", &data_tmp);
          tow_burn_turn_coeff.side_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_burn_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                            tow_burn_turn_coeff.side_coeff[i], descript););
    }

    for (i=0; i <= data_tmp_int; i++) {
          fscanf(fp,"%f", &data_tmp);
          tow_burn_turn_coeff.up_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_burn_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                            tow_burn_turn_coeff.up_coeff[i], descript););
    }

    for (i=0; i <= data_tmp_int; i++) {
          fscanf(fp,"%f", &data_tmp);
          tow_burn_turn_coeff.down_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_burn_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                            tow_burn_turn_coeff.down_coeff[i], descript););
```

APPENDIX L - miss_tow.c

```
    }

    fclose(fp);
/*  END DEFAULT BURN TURN DATA FOR miss_tow.c READ FROM FILE        */

/*  DEFAULT COAST TURN DATA FOR miss_tow.c READ FROM FILE           */
    fp = fopen("/simnet/data/ms_tw_ct.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_tw_ct.d\n");
          exit();
    }

    rewind(fp);

    /*    Read degree of polynomial */

    fscanf(fp,"%d", &data_tmp_int);
    TOW_COAST_TURN_DEG = data_tmp_int;
    tow_coast_turn_coeff.deg = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("tow_miss_poly_deg(3) is%3d %s", TOW_COAST_TURN_DEG,
                descript););

    /*    Read array data    */

    for (i=0; i <= data_tmp_int; i++) {
          fscanf(fp,"%f", &data_tmp);
          tow_coast_turn_coeff.side_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_coast_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                      tow_coast_turn_coeff.side_coeff[i], descript););
    }

    for (i=0; i <= data_tmp_int; i++) {
          fscanf(fp,"%f", &data_tmp);
          tow_coast_turn_coeff.up_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_coast_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                      tow_coast_turn_coeff.up_coeff[i], descript););
    }

    for (i=0; i <= data_tmp_int; i++) {
          fscanf(fp,"%f", &data_tmp);
          tow_coast_turn_coeff.down_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("tow_coast_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                      tow_coast_turn_coeff.down_coeff[i], descript););
    }

    fclose(fp);
/*  END DEFAULT COAST TURN DATA FOR miss_tow.c READ FROM FILE       */

    tptr->mptr.state = FALSE;
    tptr->mptr.max_flight_time = TOW_MAX_FLIGHT_TIME;
    tptr->mptr.max_turn_directions = 3;
```

APPENDIX L - miss_tow.c

```
    speed_factor = MISSILE_US_SPEED_FACTOR;
    max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
    max_range_squared = max_range_limit * max_range_limit;
    tow_ammo_type = munition_US_TOW;
}


void missile_tow_set_speed_factor( scale_speed )
REAL scale_speed;
{
    speed_factor = scale_speed;
}

void missile_tow_set_max_range_limit( limit_range )
REAL limit_range;
{
    max_range_limit = limit_range;
    max_range_squared = max_range_limit * max_range_limit;
}

void missile_tow_set_ammo_type( ammo )
ObjectType ammo;
{
    tow_ammo_type = ammo;
}

/******************************************************************
 *                                                             *
 * ROUTINE: missile_tow_fire                                   *
 * PARAMETERS:    tptr - A pointer to the TOW missile to be *
 *                 fired.                                    *
 * PARAMETERS:    launch_point - The location in world                  *
 *                         coordinates that the missile is  *
 *                         launched from.                   *
 *          loc_sight_to_world - The sight to world          *
 *                               transformation matrix used *
 *                               only in this routine.      *
 *          launch_speed - The speed of the launch           *
 *                         platform (assumed to be in the    *
 *                         direction of the missile). *
 *          tube - The tube the missile was launched from.  *
 * RETURNS: none                                             *
 * PURPOSE: This routine performs the functions          *
 *          specifically related to the firing of a TOW     *
 *          missile.                                         *
 *                                                           *
 ******************************************************************/

TOW_MISSILE *missile_tow_fire (tptr, launch_point, loc_sight_to_world,
        launch_speed, tube)
TOW_MISSILE *tptr;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
int tube;
```

```
{
   MISSILE *mptr;          /* Pointer to the particular generic missile
                              pointed at by _tptr_. */

/*/
 *  Find _mptr_.
/*/
   mptr = &(tptr->mptr);
/*/
 *  Set the initial time, location, orientation, and speed of the generic
 *  missile.
/*/
   mptr->time = 0.0;
   vec_copy (launch_point, mptr->location);
   mat_copy (loc_sight_to_world, mptr->orientation);
   mptr->speed = launch_speed +
       (speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                                       tow_burn_speed_coeff, 0.0));

   mptr->init_speed = launch_speed;
/*/
 *  Set the wire as uncut.
/*/
   tptr->wire_is_cut = FALSE;
/*/
 *  Tell the rest of the world about the firing of the missile.  If this
 *  cannot be done, return.
/*/
   if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (tow_ammo_type),
            tow_ammo_type, tow_ammo_type, NULL, targetUnknown,
            objectIrrelevant, tube))
       return;
/*/
 *  If all was successful, set the missile state to TRUE and return.
/*/
   mptr->state = TRUE;
   return;
}

/*****************************************************************
 *                                                               *
 * ROUTINE: missile_tow_fly                                      *
 * PARAMETERS:    tptr - A pointer to the TOW missile that is to *
 *                  be flown out.                                *
 *          sight_location - The location in world               *
 *                           coordinates of the gunner's         *
 *                           sight.                              *
 *          loc_sight_to_world - The sight to world              *
 *                               transformation matrix used      *
 *                               only in this routine.           *
 * RETURNS: none                                                 *
 * PURPOSE: This routine performs the functions                  *
 *          specifically related to the flying a TOW             *
 *          missile.                                             *
 *                                                               *
 *****************************************************************/
```

APPENDIX L - miss_tow.c

```
void missile_tow_fly (tptr, sight_location, loc_sight_to_world)
TOW_MISSILE *tptr;
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
{
    MISSILE *mptr;          /* A pointer to the generic aspects of _tptr_. */
    REAL time;              /* The current time after launch (ticks). */
/*/
 *  Set _mptr_ and _time_.  These values are created mostly for increased
 *  readablity.
/*/
    mptr = &(tptr->mptr);
    time = mptr->time;
/*/
 *  If the missile has reached its maximum range (not the maximum distance
 *  its allowed to fly), cut the wire.
/*/
#ifdef notdeff
    if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)
        tptr->wire_is_cut = TRUE;
#endif
    if (!tptr->wire_is_cut  &&
        ((time > TOW_RANGE_LIMIT_TIME)  ||
         (max_range_limit > 0  &&
          kinematics_range_squared (veh_kinematics, mptr->location)  >
          max_range_squared) ))
        tptr->wire_is_cut = TRUE;
/*/
 *  Find the current missile speed and the cosines of the maximum allowed turn
 *  angles in each direction.  The equations used are different before and
 *  after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)
    {
        mptr->speed = mptr->init_speed +
            (speed_factor *
             missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                                     tow_burn_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = mptr->init_speed +
            (speed_factor *
             missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                                     tow_coast_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
    }
/*/
 *  If the wire has been cut, set the ground as the target; otherwise,
 *  find a target point which will fly the missile along the gunner's line of
 *  sight.  This targeting scheme takes into account the errors introduced by
 *  attempting to guide the missile in a canted position.
/*/
```

```
    if (tptr->wire_is_cut)
        missile_target_ground (mptr);
    else
        missile_target_level_los (mptr, sight_location, loc_sight_to_world);
/*/
 *  Try to actually fly the missile.  If this fails stop the missile altogether
 *  and return.
/*/
    if (!missile_util_flyout (mptr))
    {
        missile_tow_stop (tptr);
        return;
    }
    else
    {
/*/
 *      If the missile successfully flew, check for an intersection with the
 *      ground or a vehicle.  If one is found, blow up the missile, stop its
 *      flyout and return.
/*/
        if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
        {
            missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
            missile_tow_stop (tptr);
            return;
        }
    }
/*/
 *  If the missile is to continue to fly, return.
/*/
    return;
}

/*****************************************************************
 *                                                               *
 * ROUTINE: missile_tow_stop                                     *
 * PARAMETERS:    tptr - A pointer to the TOW missile that is to  *
 *                 be stopped.                                   *
 * RETURNS: none                                                 *
 * PURPOSE: This routine causes all concerned to forget          *
 *          about the missile.  It should be called when         *
 *          the flyout of any TOW missile is stopped             *
 *          (whether or not it has exploded).  Note that         *
 *          this routine can only be called within this          *
 *          module.                                              *
 *                                                               *
 *****************************************************************/

static void missile_tow_stop (tptr)
TOW_MISSILE *tptr;
{
/*/
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
/*/
```

APPENDIX L - miss_tow.c

```
    missile_util_comm_stop_missile (&(tptr->mptr), MSL_TYPE_MISSILE);
    tptr->mptr.state = FALSE;
}

/*****************************************************************
 *                                                   *
 * ROUTINE: missile_tow_cut_wire                            *
 * PARAMETERS:    tptr - A pointer to the TOW missile whose wire   *
 *          is to be cut.                                   *
 * RETURNS: none                                      *
 * PURPOSE: This routine sets a flag indicating that the      *
 *          guidance wire of this missile is cut.             *
 *                                                   *
 *****************************************************************/

void missile_tow_cut_wire (tptr)
TOW_MISSILE *tptr;
{
/*/
 *  If the the wire is not already cut, cut the wire.
/*/
    if (!tptr->wire_is_cut)
        tptr->wire_is_cut = TRUE;
}
```

Appendix M - Source code listing for rkt_hydra.c.

The following appendix contains the source code listing for rkt_hydra.c for convenience in document maintenance and understanding of the CSU.

APPENDIX M - rkt_hydra.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/rkt_hydr
a.c,v 1.4 1993/01/28 23:27:59 cm-adst Exp $ */
/*
 * $Log: rkt_hydra.c,v $
 * Revision 1.4  1993/01/28  23:27:59  cm-adst
 * P. DesMeules's changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:19:06  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/rkt_hydra.c,v 1.4 1993/01/28 23:27:59 cm-adst Exp $";

/*****************************************************************************
 *
 * Revisions:
 *
 *'    Version        Date      Author       Title                  SP/CR Number
 *     -------        --------  ---------    ------------------     ------------
 *
 *     1.2            10/23/92  R. Branson   Data File Initiali-
 *                                             zation
 *     1.3            10/30/92  R. Branson   Added pathname to data
 *                                             directory
 *     1.4            11/25/92  R. Branson   Changed %i to %d
 *
 *     1.5            01/19/93  P.Desmeules  Increased the size of the      31
 *                                             fgets to make sure the
 *                                             whole line is read in.
 ****************************************************************************/

/*****************************************************************************
 *
 *       SP/CR No.       Description of Modification
 *       ---------       ----------------------------------------------------
 *
 *                       Hard coded defines changed to array elements.
 *                       Characteristics/parameter data array added.
 *                       Added file reads for rocket characteristics/
 *                          parameters.
 *
 *                       Added "/simnet/data/" to each data file pathname.
 *
 ****************************************************************************/

/******************************************************************
 *                                                                *
 * FILE:       rkt_hydra.c                                        *
 * AUTHOR:     Kris Bartol                                        *
 * MAINTAINER: Kris Bartol                                        *
 *                                                                *
```

APPENDIX M - rkt_hydra.c

```
* PURPOSE:      This file contains routines which govern       *
*               the behavior of an Hydra70 Rocket flown with   *
*               a ballistic trajectory.                        *
* HISTORY:      10/06/90 kris                                  *
*                                                              *
* Copyright (c) 1989 BBN Systems and Technologies, Inc.       *
* All rights reserved.                                         *
*                                                              *
***************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"

#include "librva.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
#include "libmissile.h"

#include "rkt_hydra.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)        a
#else
#define P(a)
#endif

#define DEBUG   0                   /* debugging is ON */

#define HYDRA_TRAJ_FILE             "/simnet/data/hydra70.sd"
#define HYDRA_PARAM_FILE            "/simnet/data/hydra70.sp"

/*-- Define rocket performance characteristics --*/
#define HYDRA_MIN_RANGE         rkt_hydra_char[ 7]
#define HYDRA_MAX_RANGE_S5      rkt_hydra_char[ 8]
#define HYDRA_MAX_RANGE_M151    rkt_hydra_char[ 9]
#define HYDRA_MAX_RANGE_M261    rkt_hydra_char[10]
#define HYDRA_MAX_RANGE_M255    rkt_hydra_char[11]

/*-- Define the states of an HYDRA70_ROCKET --*/
#define HYDRA_FREE      0       /* Rocket available to launch */
#define HYDRA_FLY       1       /* Rocket flying */
#define HYDRA_DETONATE  2       /* Rocket detonates - release or impact */
#define HYDRA_FALL      3       /* Sub-munitions falling.....           */
#define HYDRA_RELEASED  4       /* Sub-munitions released towards impact */
#define HYDRA_REMOVE    10      /* Rocket gets killed at end of this tick */
```

APPENDIX M - rkt_hydra.c

```
static REAL rkt_hydra_char[12] =
{
  M151_BURST_SPREAD,      /* twin bursts are 3 m apart     */
  M261_BURST_HEIGHT,      /* release submunitions 180 ft    */
  M261_BURST_RANGE,       /* 0 m in front of target (49 ?) */
  M261_BURST_SPREAD,      /* twin bursts are 13 m apart    */
  M255_BURST_RANGE,       /* release darts 150 m front of tgt */
  M255_BURST_SPREAD,      /* twin bursts are 35 m apart       */
  FLECH_60_MAX_RANGE,     /* darts fly total of 750 m         */
    50.0,            /* hydra minimum range */
  5000.0,            /* hydra maximum range for Soviet S-5 57mm Rocket */
  7000.0,            /* hydra maximum range for _M151 [actual 9000 m] */
  7000.0,            /* hydra maximum range for M261 */
  3200.0             /* hydra maximum range for M255  */
};

/*-- burst releases 9 bombletts    --*/
static int m73_per_m261_burst = M73_PER_M261_BURST;

/*-- pointer_to & number_of HYDRA70_ROCKET array --*/
static HYDRA_ROCKET *hydra_array;       /* A pointer to Hydra70_Rkt memory */
static int num_hydra;                   /* The number of defined missiles */

/*-- array of pointers to Hydra70_Rockets in flight --*/
static HYDRA_ROCKET *hydra_fly[MAX_HYDRA70_ROCKET];
static int rkts_in_flight;

/*-- Ballistics Table ... array of structures _MISSILE_BALLISTIC_OFFSETS_ --*/
static MISSILE_BALLISTIC_OFFSETS ball_table[BALLISTIC_TABLE_SIZE];
static int table_size;
static BOOLEAN ball_table_loaded = FALSE;

static VehicleID null_vehicleID;
static int flight_time; /* Time Of Flight for ballistic traj   */
static REAL
    max_range_limit,    /* [ MISSILE_US_MAX_RANGE_LIMIT ]                  */
    speed_factor,       /* [ MISSILE_US_SPEED_FACTOR ]                     */
    pylon_x,            /* [0.0] <xyz> position offset of pylon            */
    pylon_y,            /* [0.0] */
    pylon_z;            /* [0.0] */
static int flechette_veh_list;  /* list ID of flechette target vehicles */

static void missile_hydra_stop ();
static void missile_hydra_purge_free_missiles ();


/****************************************************************
 *                                                             *
 * ROUTINE: missile_hydra_init                                 *
 * PARAMETERS:    rocket_array - Array of rockets of structure *
 *                       type _HYDRA_ROCKET_                   *
 *             num_rockets - The number rockets defined in     *
 *                           _rockets_array_.                  *
 * RETURNS: none                                               *
```

APPENDIX M - rkt_hydra.c

```
 * PURPOSE: This routine copies the parameters into              *
 *          variables static to this module and initializes *
 *          the state of all the rockets.                    *
 *                                                           *
 ****************************************************************/

void missile_hydra_init( rocket_array, num_rocket )
HYDRA_ROCKET *rocket_array;
int num_rocket;
{
      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ HYDRA rocket file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR rkt_hydra.c READ FROM FILE              */
      fp = fopen("/simnet/data/rkt_hydr.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rkt_hydr.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */

      fscanf(fp,"%d", &data_tmp_int);
      m73_per_m261_burst = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("m73_per_m261_burst is%3d %s", m73_per_m261_burst,
             descript););

      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            rkt_hydra_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("rkt_hydra_char(%3d) is%11.3f %s", i,
                   rkt_hydra_char[i], descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR rkt_hydra.c READ FROM FILE          */

    hydra_array = rocket_array;
    num_hydra = num_rocket < MAX_HYDRA70_ROCKET ?
        num_rocket : MAX_HYDRA70_ROCKET;
    for (i = 0; i < MAX_HYDRA70_ROCKET; i++)
    {
        hydra_array[i].bmptr.state = HYDRA_FREE;
        hydra_array[i].bmptr.missile_id = 0;
```

```
    }
    rkts_in_flight = 0;  /* no missiles in flight */
    for( i = 0; i < MAX_HYDRA70_ROCKET; i++ )
        hydra_fly[i] = 0;

    pylon_x = 0.0;
    pylon_y = 0.0;
    pylon_z = 0.0;
    flight_time = 0;
    speed_factor = MISSILE_US_SPEED_FACTOR;
    max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;

    if (!ball_table_loaded)
      {
/*
 * load Hydra70 Rocket's ballistic table
 */
    printf( "loading Hydra70 Rocket's ballistic table  %s\n",
            HYDRA_TRAJ_FILE );
    table_size =
        missile_util_load_ball_traj_file( HYDRA_TRAJ_FILE, ball_table );
      ball_table_loaded = TRUE;
      }

/*
 * create _flechette_veh_list_ for proximity fuze
 */
    flechette_veh_list = rva_create_output_list( flechette_is_valid_veh );
#ifdef notdef
    flechette_veh_list = RVA_ALL_VEHICLES_LIST;
#endif
/*
 * initialize the proximity fuze for rockets armed with Flechette's
 */
    missile_fuze_prox_init();
}

int missile_hydra_is_free( rocket )
int rocket;
{
    return( (hydra_array[rocket].bmptr.state == HYDRA_FREE ));
}


/*****************************************************************
 * ROUTINE: missile_hydra_set_pylon_position_offsets            *
 * PARAMETERS:     x = X offset (in meters )from center of HULL.  *
 *                y = Y offset.                                 *
 *                z = Z offset.                                 *
 * RETURNS:       none.                                         *
 * PURPOSE:       Sets the X, Y and Z offsets from center of    *
 *                HULL for trajectory calculations.             *
 *****************************************************************/
void missile_hydra_set_pylon_position_offsets( x, y, z )
REAL x, y, z;
```

APPENDIX M - rkt_hydra.c

```
{
    pylon_x = x;
    pylon_y = y;
    pylon_z = z;
}

void missile_hydra_set_speed_factor( speed_scale )
REAL speed_scale;
{
    speed_factor = speed_scale;
}

void missile_hydra_set_max_range_limit( limit_range )
REAL limit_range;
{
    max_range_limit = limit_range;
}

/*****************************************************************
 * ROUTINE: missile_hydra_set_pylon_articulation                  *
 * PARAMETERS:    tgt_range - Range to target.                      *
 *               rkt_type - Type of Rocket to be launched.          *
 *               time - Pointer to Time Of Flight                   *
 *                       variable in vehicle-spec code. [int]       *
 *               se_angle - Pointer to Super Elevation              *
 *                       variable in vehicle-spec code. [REAL]      *
 *               lead_angle - Pointer to Lead Elevation             *
 *                       variable in vehicle-spec code. [REAL]      *
 *                                                                  *
 * RETURNS:      none.                                              *
 * PURPOSE:      Sets _laser_range_ of next Hydra70 rocket to       *
 *               be launched and calculates Time Of Flight,         *
 *               Super Elevation angle and Lead angle for next      *
 *               rocket launch.                                     *
 *****************************************************************/

void missile_hydra_set_pylon_articulation( tgt_range, rkt_type, time,
                                           se_angle, lead_angle )

REAL    tgt_range;
int     rkt_type, *time;
REAL    *se_angle, *lead_angle;
{
    REAL range;          /*  Range to target */
    REAL ball_range;     /*  Range to look-up in Ballistic Table */

    if( tgt_range < HYDRA_MIN_RANGE )
        range = HYDRA_MIN_RANGE;
    else if(( max_range_limit > 0.0 )  &&
            ( tgt_range > max_range_limit ) )
        range = max_range_limit;
    else
        range = tgt_range;
/* SuperElevation & TOF  for each Rocket Type */
    switch( rkt_type )
    {
```

APPENDIX M - rkt_hydra.c

```
  case ROCKET_HE:                                 /* type 10lb WARHEAD */
      if( range > HYDRA_MAX_RANGE_M151 )
          range = HYDRA_MAX_RANGE_M151;
      ball_range = range / speed_factor;
      missile_util_ballistics_calc_traj( ball_table, table_size,
                                        ball_range, 0.0, 0.0,
                                        time, se_angle );

      *lead_angle = atan( (rkt_hydra_char[ 0] - pylon_x) / range );
      *time = -5;         /* Does not have a timed fuze */
      break;
  case ROCKET_MPSM:                               /* type MPSM */
      if( range > HYDRA_MAX_RANGE_M261 )
          range = HYDRA_MAX_RANGE_M261;
      ball_range = range / speed_factor;
      missile_util_ballistics_calc_traj( ball_table, table_size,
                                        ball_range, 0.0, rkt_hydra_char[ 1],
                                        time, se_angle );

      *lead_angle = atan( (rkt_hydra_char[ 3] - pylon_x) / range );
      break;
  case ROCKET_FLECHETTE:                          /* type FLECHETTE */
      if( range > HYDRA_MAX_RANGE_M255 )
          range = HYDRA_MAX_RANGE_M255;
      ball_range = range / speed_factor;
      missile_util_ballistics_calc_traj( ball_table, table_size,
                                        ball_range, rkt_hydra_char[ 4], 0.0,
                                        time, se_angle );

      *lead_angle = atan((rkt_hydra_char[ 5] - pylon_x) /
                         (range - rkt_hydra_char[ 4]));

      break;
  default:
      printf( "hydra_set_pylon_articul: unknown warhead_type %d\n", rkt_type );
      *time = 0;
      *se_angle = 0.0;
      *lead_angle = 0.0;
      break;
  }
  flight_time = *time;
}


/*******************************************************************
 * ROUTINE: missile_hydra_fire                                      *
 * PARAMETERS:    rkt_type - Type of Rocket warhead.                  *
 *                ammo - Ammo Type of rocket's warhead.             *
 *                launch_pt - The location in world                 *
 *                            coordinates that the rocket is        *
 *                            launched from.                        *
 *            launch_orient - The sight to world                    *
 *                            transformation matrix used            *
 *                            only in this routine.                 *
 *                launch_speed - Speed of launch platform           *
 *                               (assumed to be in the direction    *
 *                               of the Rocket).                    *
 *                                                                  *
 * RETURNS: TRUE if successful, FALSE if not.                       *
```

APPENDIX M - rkt_hydra.c

```
 * PURPOSE: This routine performs the functions                  *
 *          specifically related to the firing of a HYDRA70 *
 *          rocket.                                              *
 *****************************************************************/

int missile_hydra_fire( rkt_type, ammo, launch_pt,
                        launch_orient, launch_speed )

int rkt_type;
ObjectType ammo;
VECTOR launch_pt;
T_MAT_PTR launch_orient;
REAL launch_speed;
{
    T_MATRIX
        launch_lead,
        launch_se;
    REAL
        se_angle,        /* munition_specific SuperElevation angle */
        lead_angle;      /* munition_specific (+/-)Lead angle */
    int time;            /* munition_specific FlightTime */
    HYDRA_ROCKET *rkt;
    BALLISTIC_MISSILE *bmptr;
    ObjectType fuze;
    int i, valid_msl;

/* get next FREE rocket */
    valid_msl = 0;
    rkt = hydra_array;
    for( i = 0; i < MAX_HYDRA70_ROCKET; i++, rkt++ )
        if( rkt->bmptr.state == HYDRA_FREE )
        {
            valid_msl = 1;
            hydra_fly[rkts_in_flight] = rkt;
            bmptr = &(rkt->bmptr);
#if DEBUG
            printf( "Launching Rocket %d\n", i );
#endif
            rkts_in_flight++;            /* rkts_in_flight == # flying */
            break;
        }
    if( !valid_msl )            /* no available missile to launch */
    {
        return( FALSE );
    }

/* set MaxRange for Rocket Type */
    switch( rkt_type )
    {
    case ROCKET_HE:                       /* High Explosive */
        bmptr->max_range = HYDRA_MAX_RANGE_M151;
        rkt->sub_mun_type = SUB_MUN_NONE;
        rkt->sub_ammo_type = 0;
        fuze = munition_US_M433;
        break;
    case ROCKET_MPSM:                     /* Multi-Purpose Sub-Munition */
```

APPENDIX M - rkt_hydra.c

```
        bmptr->max_range = HYDRA_MAX_RANGE_M261;
        rkt->sub_mun_type = SUB_MUN_IMPACT;
        rkt->sub_ammo_type = munition_US_M73;
        rkt->sub_munition.impact.ammo = munition_US_M73;
        rkt->sub_munition.impact.fuze = munition_US_M433;
        rkt->sub_munition.impact.quantity = m73_per_m261_burst;
        rkt->sub_munition.impact.height = rkt_hydra_char[ 1];
        fuze = munition_US_M439;
        break;
    case ROCKET_FLECHETTE:              /* Flechette discharging warhead */
        bmptr->max_range = HYDRA_MAX_RANGE_M255;
        rkt->sub_mun_type = SUB_MUN_CANISTER;
        rkt->sub_ammo_type = munition_US_Flechette_60;
        rkt->sub_munition.dart.ammo = munition_US_Flechette_60;
        rkt->sub_munition.dart.fuze = 0;
        fuze = munition_US_M439;
        break;
    default:
        printf( "hydra_fire_rkt: unknown rocket_type %d\n", rkt_type );
        rkts_in_flight--;
        bmptr -> state = HYDRA_FLY;
        return( FALSE );
        break;
    }
    mat_copy( launch_orient, bmptr->launcher_C_world );
    mat_copy( launch_orient, bmptr->orientation );
    vec_copy( launch_pt, bmptr->location );
    bmptr->speed = launch_speed;
/* -- Tell the rest of the world about the firing of this B-missile. --
 * -- If this cannot be done, return FALSE. --
 */

    if( !missile_util_comm_fire_missile
        ( bmptr, MSL_TYPE_BALLISTIC,
        map_get_ammo_entry_from_network_type( ammo ),
        ammo, ammo,/*guises*/
        &(null_vehicleID), 0/*targ_type*/, fuze, 0/*tube*/ ))
    {
        rkts_in_flight--;
        bmptr -> state = HYDRA_FLY;
        return( FALSE );
    }

    bmptr -> max_flight_time = flight_time;
    bmptr -> ammo_type = ammo;
    bmptr -> time = 0;                  /* initialize in-flight timer */
    bmptr -> ball_index = 0;            /* first point into Ball-table */
    bmptr -> state = HYDRA_FLY;         /* rocket is now flying */
    return( TRUE );
}

/*******************************************************************
 * ROUTINE:     missile_hydra_fly_rockets                          *
 * PARAMETERS:    none                                               *
 * RETURNS:     none                                               *
```

APPENDIX M - rkt_hydra.c

```
 * PURPOSE: This routine flys out all rockets that are in      *
 *               a flying state.                                *
 ****************************************************************/

void missile_hydra_fly_rockets()
{
    register int i;
    int at_least_one_empty_MPSM;

/*      Fly out all launched & flying rockets.
 *      -- may have to also 'fly out' all released submunitions --
 */
    at_least_one_empty_MPSM = FALSE;
    for( i = 0; i < rkts_in_flight; i++ )
    {
        switch( hydra_fly[i]->bmptr.state )
        {
        case HYDRA_FREE:
            hydra_fly[i]->bmptr.state = HYDRA_REMOVE;
            break;
        case HYDRA_FLY:
            missile_hydra_fly( hydra_fly[i] );
            break;
        case HYDRA_DETONATE:
            switch( hydra_fly[i]->sub_ammo_type )
            {
            case munition_US_M73:                   /* MPSM bomblets */
                missile_m73_init
                    ( &(hydra_fly[i]->bmptr),
                     &(hydra_fly[i]->sub_munition),
                       ball_table[ hydra_fly[i]->bmptr.ball_index ].speed );
                hydra_fly[i]->bmptr.state = HYDRA_FALL;
                break;
            case munition_US_Flechette_60:          /* FLECHETTE darts */
                missile_flechette_init
                    ( &(hydra_fly[i]->bmptr),
                     &(hydra_fly[i]->sub_munition),
                     ball_table[ hydra_fly[i]->bmptr.ball_index ].speed );
                hydra_fly[i]->bmptr.state = HYDRA_RELEASED;
                break;
            default:
                printf( "Hydra_Detonate: R_%d unknown ammo-type\n",i );
                missile_hydra_stop( hydra_fly[i] );
                break;
            }
            break;
        case HYDRA_FALL:
            switch( hydra_fly[i]->sub_ammo_type )
            {
            case munition_US_M73:                   /* type MPSM */
                if( missile_m73_drop( &(hydra_fly[i]->bmptr),
                                      &(hydra_fly[i]->sub_munition) ))
                    hydra_fly[i]->bmptr.state = HYDRA_RELEASED;
                break;
            default:
```

APPENDIX M - rkt_hydra.c

```
                printf( "Hydra_Fall(): R_%d bad sub_munition\n",i );
                missile_hydra_stop( hydra_fly[i] );
                break;
            }
            break;
        case HYDRA_RELEASED:
            switch( hydra_fly[i]->sub_ammo_type )
            {
            case munition_US_M73:                  /* type MPSM */
                if( ! missile_m73_impact( &(hydra_fly[i]->bmptr),
                                          &(hydra_fly[i]->sub_munition) ))
                {
                    at_least_one_empty_MPSM = TRUE;
                    missile_hydra_stop( hydra_fly[i] );
                }
                break;
            case munition_US_Flechette_60:         /* type FLECHETTE */
                if( ! missile_flechette_fly( &(hydra_fly[i]->bmptr),
                                             &(hydra_fly[i]->sub_munition),
                                             flechette_veh_list ))
                {
                    missile_hydra_stop( hydra_fly[i] );
                    missile_fuze_prox_stop
                        ( &(hydra_fly[i]->sub_munition.dart.pptr) );
                }
                break;
            default:
                printf( "Hydra_Release: R_%d bad sub_munition\n",i );
                missile_hydra_stop( hydra_fly[i] );
                break;
            }
            break;
        case HYDRA_REMOVE:
            break;
        default:
            printf( "Msl_hydra_fly_rkts(): rkt_%d not flying\n", i );
            missile_hydra_stop( hydra_fly[i] );
            break;
        }
    }
/* Send out remaining (if any) Indirect Fire pkts */
    if( at_least_one_empty_MPSM )
        network_ifire_send_indirect_fire();

/* Get rid of DEAD rockets */
    missile_hydra_purge_free_missiles();
}


/******************************************************************
 * ROUTINE:     missile_hydra_fly                                 *
 * PARAMETERS:    rkt - Pointer to a _HYDRA_ROCKET_ structure       *
 * RETURNS:     none                                              *
 * PURPOSE: This routine performs the functions                 *
 *          specifically related to the flying an HYDRA70       *
```

```
 *              and frees up the Rocket for another launch.      *
 ***************************************************************/

static void missile_hydra_stop( rkt )
HYDRA_ROCKET *rkt;
{
    BALLISTIC_MISSILE    *bmptr;
    int i;

    bmptr = &( rkt->bmptr );
/*
 *  Tell the world to stop worrying about this missile then release the
 *  memory for use by other missiles.
 */
    missile_util_comm_stop_missile( bmptr, MSL_TYPE_BALLISTIC );

#if DEBUG
    printf( "stop:: T: %d   Rkt: %d  Pos: %1.21f %1.21f %1.21f\n",
            bmptr->time, bmptr->missile_id, bmptr->location[0],
            bmptr->location[1], bmptr->location[2] );
#endif
/*
 * Mark rocket to be Removed
 */
    bmptr->state = HYDRA_REMOVE;
}

static void missile_hydra_purge_free_missiles()
{
    int i;

    i = 0;
    while( i < rkts_in_flight )
    {
        if( hydra_fly[i]->bmptr.state == HYDRA_REMOVE )
        {
            /*
             *  Swap --BAD-- rocket[i] with --LAST-- rocket[rkts_in_flight]
             *  Cut-off (now BAD) --LAST-- rocket
             *  Check (now Good) rocket[i]
             */
            hydra_fly[i]->bmptr.state = HYDRA_FREE;
            rkts_in_flight--;
            hydra_fly[i] = hydra_fly[rkts_in_flight];
            hydra_fly[rkts_in_flight] = 0;
        }
        else
            /*
             *  Check next rocket[i+1]
             */
            i++;
    }
}
```

APPENDIX M - rkt_hydra.c

```
void mbmat( mat )
T_MAT_PTR mat;
{
    int i, j;
    for( i=0; i<3; i++ )
    {
        for( j=0; j<3; j++ )
            printf( "  %1.41f ", mat[i][j] );
        printf( "\n" );
    }
}

void mbmat_nan( mat )
T_MAT_PTR mat;
{
    int i, j;
    union foo
    {
        REAL df;
        long l[2];
    } x;

    for( i=0; i<3; i++ )
    {
        for( j=0; j<3; j++ )
            printf( "  %1.41f ", mat[i][j] );
        printf( "-->" );
        for( j=0; j<3; j++ )
        {
            x.df = mat[i][j];
            printf( "  0x%08x 0x%08x", x.l[0], x.l[1] );
        }
        printf( "\n" );
    }
}

void mbm( n, msg )
int n;
char msg[];
{
    printf( "BM: %d -> %s\n", n, msg );
}

void mbfl( n, msg )
REAL n;
char msg[];
{
    printf( "BM: %6.41f -> %s\n", n, msg );
}
```

Appendix N - Source code listing for rwa_hydra.c.

The following appendix contains the source code listing for rwa_hydra.c for convenience in document maintenance and understanding of the CSU.

APPENDIX N - rwa_hydra.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_hydra.c,v 1.4
1993/01/28 23:33:00 cm-adst Exp $ */
/*
 * $Log: rwa_hydra.c,v $
 * Revision 1.4  1993/01/28  23:33:00  cm-adst
 * P. DesMeueles's changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:29:20  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  17:02:58  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/R
CS/rwa_hydra.c,v 1.4 1993/01/28 23:33:00 cm-adst Exp $";

/*************************************************************************
 *
 * Revisions:
 *
 *'    Version        Date        Author       Title                    SP/CR Number
 *     _______        ________    __________   ____________________     ____________
 *
 *     1.2            10/23/92    R. Branson   Data File Initiali-
 *                                               zation
 *     1.3            10/30/92    R. Branson   Added pathname to data
 *                                               directory
 *
 *     1.5            01/19/93    P.Desmeules  Increased the size of the     31
 *                                               fgets to make sure the
 *                                               whole line is read in.
*************************************************************************/

/*************************************************************************
 *
 *      SP/CR No.        Description of Modification
 *      __________       ____________________________________________
 *
 *                       Hard coded defines changed to array elements.
 *                       Characteristics/parameter data array added.
 *                       Added file reads for hydra rocket characteristics/
 *                          parameters.
 *
 *                       Added "/simnet/data/" to each data file pathname.
 *
*************************************************************************/

/*****************************************************************
 * SYSTEM NAME:  rwa                                             *
 * FILE:         rwa_hydra.c                                     *
 * AUTHOR:       Kris Bartol                                     *
 *                                                               *
 * SIMNET simulation of Hydra70 Rocket                           *
 *
```

APPENDIX N - rwa_hydra.c

```
 * Copyright (c) 1990 BBN Advanced Simulation Division.              *
 * All rights reserved.                                              *
 *                                                                   *
 *****************************************************************/
#include "simstdio.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "sim_macros.h"
#include "basic.h"
#include "mun_type.h"
#include "veh_type.h"

#include "libmatrix.h"
#include "libmath.h"
#include "librotate.h"
#include "libturret.h"
#include "libhull.h"
#include "libkin.h"
#include "libcig.h"
#include "libimps.h"
#include "libmap.h"
#include "libmissile.h"
#include "libmiss_dfn.h"
#include "rkt_hydra.h"

#include "rwa_kinemat.h"
#include "rwa_weapons.h"
#include "rwa_meter.h"
#include "rwa_config.h"


/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

#define DEBUG   0       /* debugging is ON */

#define LEFT    0
#define RIGHT   1

#define NUM_ROCKETS_LAUNCHED_PER_TICK   2

/*/
 *  Define rocket characteristics.
/*/

#define HYDRA_LAUNCHER_POS_X  hydra_rkt_char[0]
#define HYDRA_LAUNCHER_POS_Y  hydra_rkt_char[1]
#define HYDRA_LAUNCHER_POS_Z  hydra_rkt_char[2]
```

APPENDIX N - rwa_hydra.c

```
/* *****
 * Articulation Limits are +4 to -15 degrees but are adjusted to
 * +19 to -15 degrees for simulation's fixed OTW reticle
 * *****/
#define SOVIET_ARTICULATION      ( mil_to_rad(hydra_rkt_char[3]))
#define HULL_NEG_5_PITCH         ( deg_to_rad(hydra_rkt_char[4]))
#define ARTICULATION_MAX         ( deg_to_rad(hydra_rkt_char[5]))
#define ARTICULATION_MIN         ( deg_to_rad(hydra_rkt_char[6]))

/*/
 *  Hydra rocket characteristic parameters initialized to default values.
/*/
static REAL hydra_rkt_char[7] =
{
    4.5,     /* hydra launcher position X */
    0.5,     /* hydra launcher position Y */
   -2.0,     /* hydra launcher position Z */
  104.0,     /* mils of Soviet articulation */
   -5.0,     /* degrees of hull negative pitch */
   19.0,     /* degrees of maximum articulation */
  -15.0      /* degrees of minimum articulation */
};

ROTATE_ELEMENT_DEF (articulation_element);
ROTATE_ELEMENT_DEF (pylon_L_element);
ROTATE_ELEMENT_DEF (pylon_R_element);

static HYDRA_ROCKET hydras[MAX_HYDRA70_ROCKET + 1] = { 0 };

static VehicleID null_VehicleID;
static int flight_time;          /* Time Of Flight for ballistic traj */
static REAL
    super_elevation,             /* Adj angle for ballistic traj */
    target_range;                /* Range by which to calculate ballistics */

static ObjectType ammo_type;     /* Ammo_Type of rockets to be launched */
static int warhead_class;        /* one of [ HE | MPSM | FLECHETTE ]     */

static int pylons_set;           /* TRUE when pylon articulation is complete */
static int left_rocket_launch;   /* TRUE --> launch left rocket  */
static int right_rocket_launch;  /* TRUE --> launch right rocket */

static VECTOR left_launcher_pos = { 4.5, 0.0, 0.0 };
static VECTOR right_launcher_pos = { 4.5, 0.0, 0.0 };
static VECTOR articulation_pos = { 0.0, 0.5, -2.0 };


extern REAL weapons_get_rocket_range();
extern REAL kinematics_get_true_airspeed();
extern void mbmat();
extern void mbmat_nan();
extern void mbvec();

ROTATE_ELEMENT *articulation()
{
```

```
    return( &articulation_element );
}

ROTATE_ELEMENT *pylon_L()
{
    return( &pylon_L_element );
}

ROTATE_ELEMENT *pylon_R()
{
    return( &pylon_R_element );
}


void hydra_launch_rocket_left()
{
    left_rocket_launch = TRUE;
}

void hydra_launch_rocket_right()
{
    right_rocket_launch = TRUE;
}

int hydra_launch_rocket( launch_from_right )
int launch_from_right;  /* 0 = left-side (neg) :: 1 = right-side (pos) */
{
    T_MAT_PTR launch_orient;
    VECTOR launch_velocity;
    REAL
        *launch_point,
        se_angle,
        lead_angle;

/* get launch_point  &  launch_orient */
    if( launch_from_right )      /* launch from right */
    {
        launch_point = rotate_get_loc( world(), pylon_R() );
        launch_orient = rotate_get_mat( pylon_R(), world() );
    }
    else
    {
        launch_point = rotate_get_loc( world(), pylon_L() );
        launch_orient = rotate_get_mat( pylon_L(), world() );
    }
#if DEBUG
    if( mat_check(launch_orient) == FALSE )
        mbmat_nan( launch_orient );
#endif
    if( !missile_hydra_fire( warhead_class, ammo_type,
                             launch_point, launch_orient,
                             (kinematics_get_true_airspeed()/15) /*init speed*/))

    {
#if DEBUG
        printf( "No memory in missile_comm for HYDRA\n");
```

APPENDIX N - rwa_hydra.c

```
#endif
        printf( "Rocket launch failed\n" );
        return( FALSE );
    }
    return( TRUE );
}


int hydra_pylons_are_set()
{
    return( pylons_set );
}


void hydra_set_pylon_articulation( WAS_position )
int WAS_position;
{
    MUNITION_DATA *mun_data;
    int flight_time;    /* time of flight to fly _range_ meters */
    REAL
        range,          /* range to target */
        super_elev,     /* super elevation angle for trajectory */
        dispersion;     /* dispersion angle for trajectory */
/*
 * Given _range_ & _ammo_type_ ::
 *      * calculate and return super_elev & dispersion angles
 *      * calculate and set Time-Of-Flight timer
 *      * set _ammo_type_ of next rocket(s) to be fired
 */
    mun_data = rwa_config_get_was_munition_info (WAS_position);
    ammo_type = mun_data->munition_type;

    if (mun_data->code != MUNITION_ROCKET)
      /* bombs, for example */
      return;

    switch(mun_data->data.rocket.warhead)
    {
    case WARHEAD_HE:
        warhead_class = ROCKET_HE;
        break;
    case WARHEAD_MPSM:
        warhead_class = ROCKET_MPSM;
        break;
    case WARHEAD_FLECHETTE:
        warhead_class = ROCKET_FLECHETTE;
        break;
    default:
        printf( "hydra_set_artic: unknown warhead %d for WAS %d\n",
                mun_data->data.rocket.warhead, WAS_position );
        break;
    }
  /*
   * Get rocket range & calculate SuperElevation and Dispersion angles
   */
```

APPENDIX N - rwa_hydra.c

```
    pylons_set = FALSE;
    if( mun_data->data.rocket.articulation )
        range = weapons_get_rocket_range();
    else
        range = (REAL)(mun_data->data.rocket.flyout_range);
/*
 * Set pylon Super Elevation angle  &  pylon Dispersion angle
 */
    missile_hydra_set_pylon_articulation( range, warhead_class, &flight_time,
                                          &super_elev, &dispersion );

    super_elev += HULL_NEG_5_PITCH;
    rotate_set_angle( articulation(), super_elev );
    rotate_set_angle( pylon_R(), (- dispersion) );
    rotate_set_angle( pylon_L(), dispersion );
}


void hydra_config_rockets()
{
    MUNITION_DATA *mun_data;
    int i;

    for( i = 0; i < MAX_WAS_POSITIONS; i++ )
    {
        if( (mun_data = rwa_config_get_was_munition_info( i )) == NULL )
            continue;
        if( mun_data->code == MUNITION_ROCKET )
        {
            missile_hydra_set_speed_factor
                ( (REAL)(mun_data->data.rocket.speed_factor) );
            missile_hydra_set_max_range_limit
                ( (REAL)(mun_data->data.rocket.flyout_range) );
        }
    }
}

void hydra_init ()
{
      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ HYDRA file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR rwa_hydra.c READ FROM FILE           */
      fp = fopen("/simnet/data/rwa_hydr.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rwa_hydr.d\n");
            exit();
      }

      rewind(fp);
```

APPENDIX N - rwa_hydra.c

```
    /*    Read array data    */
    i=0;

    while(fscanf(fp,"%f", &data_tmp) != EOF){
          hydra_rkt_char[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("hydra_rkt_char(%3d) is%11.3f %s", i,
                hydra_rkt_char[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR rwa_hydra.c READ FROM FILE     */

    left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
    right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
    articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
    articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;

   if(!rotate_init_element( &articulation_element, hull(),
                              1.0, 0.0, 0.0, 0.0,
                          ARTICULATION_MIN,ARTICULATION_MAX,/*TWO_*/PI,/*rate*/
                          0.0, HYDRA_LAUNCHER_POS_Y, HYDRA_LAUNCHER_POS_Z ))
   {
      printf( "Rotate_Init_Element: articulation_element FAILED\n" );
   }

   rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
                        -TWO_PI, TWO_PI, TWO_PI, /*rate*/
                        -HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
   rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
                        -TWO_PI, TWO_PI, TWO_PI, /*rate*/
                        HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
   missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
   missile_hydra_set_pylon_position_offsets( HYDRA_LAUNCHER_POS_X,
                                             HYDRA_LAUNCHER_POS_Y,
                                             HYDRA_LAUNCHER_POS_Z );

   hydra_config_rockets();
   left_rocket_launch = FALSE;
   right_rocket_launch = FALSE;
   pylons_set = FALSE;
}

void hydra_simul()
{
    missile_hydra_fly_rockets();

    if( !pylons_set )
    {
        pylons_set = TRUE;
        rotate_set_no_rotate( pylon_R() );
        rotate_set_no_rotate( pylon_L() );
        rotate_set_no_rotate( articulation() );
    }
```

```
    else
    {
        if( left_rocket_launch )
            if( hydra_launch_rocket( LEFT ) )
                left_rocket_launch = FALSE;
        if( right_rocket_launch )
            if( hydra_launch_rocket( RIGHT ) )
                right_rocket_launch = FALSE;
    }
}

void mbvec( str, vec )
char *str;
VECTOR vec;
{
    printf( "%s [ %1.4lf %1.4lf %1.4lf ]\n",
            str, vec[X], vec[Y], vec[Z] );
}
```

Appendix O - Source code listing for sub_flech.c.

The following appendix contains the source code listing for sub_flech.c for convenience in document maintenance and understanding of the CSU.

APPENDIX O - sub_flech.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/sub_flec
h.c,v 1.4 1993/01/28 23:27:09 cm-adst Exp $ */
/*
 * $Log: sub_flech.c,v $
 * Revision 1.4  1993/01/28  23:27:09  cm-adst
 * P.DesMeules's changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:19:27  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/sub_flech.c,v 1.4 1993/01/28 23:27:09 cm-adst Exp $";

/****************************************************************************
 *
 * Revisions:
 *
 *      Version         Date        Author        Title                       SP/CR Number
 *      _______         ________    _________     ____________________        ___________
 *
 *      1.2             10/23/92    R. Branson    Data File Initiali-
 *                                                  zation
 *      1.3             10/30/92    R. Branson    Added pathname to data
 *                                                  directory
 *      1.4             11/25/92    R. Branson    Changed %i to %d
 *
 *      1.5             01/19/93    P.Desmeules   Increased the size of the      31
 *                                                  fgets to make sure the
 *                                                  whole line is read in.
 ****************************************************************************/

/****************************************************************************
 *
 *        SP/CR No.        Description of Modification
 *        _________        ___________________________________________________
 *
 *                         Hard coded defines changed to array elements.
 *                         Characteristics/parameter data array added.
 *                         Added file reads for sub_flechette  characteristics/
 *                            parameters and flechette speed coefficients.
 *
 *                         Added "/simnet/data/" to each data file pathname.
 *
 ****************************************************************************/

/*****************************************************************
 *                                                               *
 * FILE:        sub_flech.c                                      *
 * AUTHOR:      Kris Bartol                                      *
 * MAINTAINER:  Kris Bartol                                      *
 *                                                               *
```

APPENDIX O - sub_flech.c

```
 * PURPOSE:       This file contains routines which simulates      *
 *                the behavior of sub-munitions of type             *
 *                munition_US_Flechette_60.                         *
 * HISTORY:       10/06/90 kris                                     *
 *                                                                  *
 * Copyright (c) 1989 BBN Systems and Technologies, Inc.            *
 * All rights reserved.                                             *
 *                                                                  *
 *****************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"

#include "libhull.h"
#include "libimps.h"
#include "libkin.h"
#include "libmath.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"

#include "rkt_hydra.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

#define DEBUG   0                    /* debugging is ON */

#define INVEST_DIST_SQ          sub_flech_char[0]
#define FUZE_DIST_SQ              sub_flech_char[1]
#define FLECHETTE_SPEED_DEG       sub_flech_poly_deg

/*/
 *  Sub_flechette characteristic parameters initialized to default values.
/*/
static REAL sub_flech_char[3] =
{
 10000.0,   /* (100 m)^2  :: max speed < 100 */
   306.25,  /* (17.5 m)^2 :: flechettes fly
                             in a cylinder with a radius
                             of 17.5 m and length of 750 m */
 FLECH_60_MAX_RANGE     /* darts fly total of 750m */
};
```

APPENDIX O - sub_flech.c

```
/*/
 *  The following term sets the order of the polynomial used to determine
 *  the speed of the flechettes.
/*/
static int sub_flech_poly_deg = 3;


/*/
 *  Coefficients for the speed polynomial for flechettes initialized
 *  to default values.
/*/

static REAL flechette_speed_coef[5] =
{
    41.75,                /* a_0 - m/tick          */
    -0.20397254,          /* a_1 - m/tick/m        */
     0.00022724278,       /* a_2 - m/tick/m^2      */
    -0.00000008633,       /* a_3 - m/tick/m^3      */
     0.0
};

static VECTOR zero_vector = { 0.0, 0.0, 0.0 };
static VehicleID null_VehicleID;


/* this routine is invoked by the rva for each vehicle to see if it
 * should be included on the flechette valid vehicle list
 */
flechette_is_valid_veh (veh)
VehicleAppearanceVariant *veh;
{
    return( /* is_alive_vehicle (veh->appearance) */ TRUE );
}


/**********************************************************************
 * ROUTINE: missile_flechette_init                                    *
 * PARAMETERS:    bmptr - Pointer to a _BALLISTIC_MISSILE_            *
 *                        structure that's ammo-type is Flechette     *
 *                        i.e. it releases sub-munitions of type      *
 *                        _munition_US_Flechette_60_.                 *
 *              sub_mun - Pointer to sub-munition structure           *
 *                        associated with _bmptr_.                    *
 *           init_speed - Terminal speed of rocket ==                 *
 *                        initial speed of flechettes.                *
 * RETURNS:       none                                                *
 * PURPOSE:       Initialize rocket's _bmptr_ to behave according     *
 *                sub-munitions type of                               *
 *                _munition_US_Flechette_60_.                         *
 **********************************************************************/

void missile_flechette_init( bmptr, sub_mun, init_speed )
BALLISTIC_MISSILE         *bmptr;
BALLISTIC_SUB_MUN         *sub_mun;
```

APPENDIX O - sub_flech.c

```
REAL                    init_speed;
{
    BALLISTIC_CANISTER *dart;
    VECTOR velocity;

      int   i;
      int   data_tmp_int;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ FLECHETTE file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR sub_flech.c READ FROM FILE          */
      fp = fopen("/simnet/data/sub_flec.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/sub_flec.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            sub_flech_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("sub_flech_char(%3d) is%11.3f %s", i, sub_flech_char[i],
                        descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR sub_flech.c READ FROM FILE      */

/*  DEFAULT FLECHETTE SPEED DATA FOR sub_flech.c READ FROM FILE          */
      fp = fopen("/simnet/data/flec_spd.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/flec_spd.d\n");
            exit();
      }

      rewind(fp);

      /*    Read degree of polynomial */

      fscanf(fp,"%d", &data_tmp_int);
      FLECHETTE_SPEED_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("sub_flech_poly_deg is%3d %s", FLECHETTE_SPEED_DEG,
                descript););

      /*    Read array data    */
      i=0;
```

APPENDIX O - sub_flech.c

```
    while(fscanf(fp,"%f", &data_tmp) != EOF){
          flechette_speed_coef[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("flechette_speed_coef(%3d) is%11.3f %s", i,
                      flechette_speed_coef[i], descript););
          ++i;
    }

    fclose(fp);
/*  END DEFAULT BURN SPEED DATA FOR sub_flech.c READ FROM FILE    */

   bmptr->time = 0;

   dart = &(sub_mun->dart);
   dart->distance = 0.0;
   dart->init_speed = init_speed;
   dart->pptr = NULL;
   vec_scale( bmptr->orientation, init_speed, velocity );
   missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                           sub_mun, SUB_MUN_CANISTER,
                                           zero_vector, velocity );
#if DEBUG
   printf( "InitSpeed %1.21f  Dist %1.21f\n", init_speed, dart->distance );
#endif
}


/*****************************************************************
 * ROUTINE: missile_flechette_fly                                *
 * PARAMETERS:    bmptr - Pointer to a _BALLISTIC_MISSILE_        *
 *                      structure that's ammo-type is Flechette *
 *                      i.e. it releases sub-munitions of type  *
 *                      _munition_US_Flechette_60_.             *
 *              sub_mun - Pointer to sub-munition structure     *
 *                      associated with _bmptr_.                *
 *              veh_list - Vehicle list ID.                     *
 * RETURNS:     none.                                           *
 * PURPOSE:     Simulates the flying of munition-type           *
 *              _munition_US_Flechette_60_.                     *
 *              ~1200 2" lead darts are released and fly a      *
 *              cylindrical pattern 35 m in diameter ...        *
 *              Hence, we simulate the flechettes with ONE      *
 *              dart flown down the center of the cylinder      *
 *              and give it a 17.5 m proximity fuze.  If the    *
 *              proximity fuze detonates, we impact the         *
 *              recipient vehicle and continue the lone dart's  *
 *              flyout to a distance of 750 m.  At this point,  *
 *              the flechette rounds have lost the momentum     *
 *              and fall to the ground -- the rocket is         *
 *              terminated.                                     *
 *****************************************************************/

int missile_flechette_fly( bmptr, sub_mun, veh_list )
BALLISTIC_MISSILE *bmptr;
```

APPENDIX O - sub_flech.c

```
BALLISTIC_SUB_MUN *sub_mun;
int veh_list;
{
    BALLISTIC_CANISTER *dart;
    VECTOR             velocity;

    dart = &(sub_mun->dart);
/*
 * SPEED */
    bmptr->speed =
        missile_util_eval_poly( FLECHETTE_SPEED_DEG, flechette_speed_coef,
                                dart->distance ) + dart->init_speed;
/*
 * DISTANCE */
    dart->distance += bmptr->speed;
    if( dart->distance >= sub_flech_char[2] )
        return( FALSE );
/*
 * VELOCITY */
    vec_scale( bmptr->orientation[Y], bmptr->speed, velocity );
/*
 * POSITION */
    vec_add( bmptr->location, velocity, bmptr->location );
/*
 * PROX_FUZE */
    if( missile_fuze_all_prox( bmptr,
                               MSL_TYPE_BALLISTIC, PROX_FUZE_ON_ALL_VEH,
                               &(null_VehicleID), &(dart->pptr),
                               veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ ) )
        do
        {
/* DETONATION ? */
            if( missile_util_comm_check_sub_mun( bmptr, MSL_TYPE_BALLISTIC,
                                                 sub_mun, SUB_MUN_CANISTER ))
                missile_util_comm_release_sub_munition( bmptr,
                                                        MSL_TYPE_BALLISTIC,
                                                        sub_mun,
                                                        SUB_MUN_CANISTER,
                                                        zero_vector,
                                                        velocity );
        } while( dart->pptr != NULL  &&
                 missile_fuze_detonate_prox( bmptr, MSL_TYPE_BALLISTIC,
                                             &(dart->pptr), FUZE_DIST_SQ, 0 ));
    return( TRUE );
}
```

Appendix P - Source code listing for sub_m73.c.

The following appendix contains the source code listing for sub_m73.c for convenience in document maintenance and understanding of the CSU.

APPENDIX P - sub_m73.c

```
/* $Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/sub_m73.c,v 1.4 1993/01/28
23:27:09 cm-adst Exp $ */
/*
 * $Log: sub_m73.c,v $
 * Revision 1.4  1993/01/28  23:27:09  cm-adst
 * P.DesMeules's changes for spcr 31
 *
 * Revision 1.3  1993/01/06  21:19:51  cm-adst
 * R.Branson's changes for the weapons model.
 *
 * Revision 1.1  1992/09/30  16:39:52  cm-adst
 * Initial Version
 *
 */
static char RCS_ID[] = "$Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/sub_m73.c,v 1.4 1993/01/28
23:27:09 cm-adst Exp $";

/*****************************************************************************
*
*, Revisions:
*
*     Version      Date         Author   Title                  SP/CR Number
*     _______     _______     _______   ________________       __________
*
*     1.2     10/23/92  R. Branson  Data File Initiali-
*                                  zation
*     1.3     10/30/92  R. Branson  Added pathname to data
*                                  directory
*     1.5     01/19/93  P.Desmeules Increased the size of the      31
*                                fgets to make sure the
*                                whole line is read in.
*
*****************************************************************************/

/*****************************************************************************
*
*       SP/CR No. Description of Modification
*       __________  _____________________________________________
*
*                 Hard coded defines changed to array elements.
*                 Characteristics/parameter data array added.
*                 Added file reads for sub_m73 characteristics/
*                    parameters.
*
*                 Added "/simnet/data/" to each data file pathname.
*
*****************************************************************************/

/*****************************************************************
 *                                                               *
 * FILE:         sub_m73.c                                       *
 * AUTHOR:       Kris Bartol                                     *
 * MAINTAINER:   Kris Bartol                                     *
```

APPENDIX P - sub_m73.c

```
 *                                                                *
 * PURPOSE:       This file contains routines which simulates     *
 *                the behavior of sub-munitions of type           *
 *                munition_US_M73.                                *
 * HISTORY:       10/06/90 kris                                   *
 *                                                                *
 * Copyright (c) 1989 BBN Systems and Technologies, Inc.          *
 * All rights reserved.                                           *
 *                                                                *
 *****************************************************************/

#include "stdio.h"
#include "math.h"

#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"

#include "libmath.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"

#include "rkt_hydra.h"

/*
* Debug macro
*/
#ifdef FILEDBG
#define P(a)      a
#else
#define P(a)
#endif

#define DEBUG   0                 /* debugging is ON */

/*/
 *  Sub M73 characteristic parameters initialized to default values.
/*/
static REAL sub_m73_char[3] =
{
  0.03266667,          /* 75% of gravity - 75% * 9.8m/sec^^2/225 ticks^^2*/
  M73_FOOT_ANGLE_X,    /* bomblettes fall w/ +/- 8.8 deg angular displ */
  M73_FOOT_ANGLE_Y     /* bomblettes fall w/ +/- 12.35 deg angular displ */
};

static REAL zero_velocity[3] = { 0.0, 0.0, 0.0 };

static void missile_m73_get_impact ();

/******************************************************************
 * ROUTINE: missile_m73_init                            *
 * PARAMETERS:    bmptr - Pointer to a _BALLISTIC_MISSILE_        *
```

APPENDIX P - sub_m73.c

```
 *                  structure that's ammo-type is MPSM        *
 *                  i.e. it releases sub-munitions of type    *
 *                  _munition_US_M73_.                        *
 *            sub_mun - Pointer to sub-munition structure     *
 *                  associated with _bmptr_.                  *
 *            speed - Terminal speed of Rocket at detonation. *
 * RETURNS:   none                                            *
 * PURPOSE:   Initialize rocket's _bmptr_ to behave according *
 *            sub-munitions type of _munition_US_M73_.        *
 ***************************************************************/

void missile_m73_init( bmptr, sub_mun, speed )
BALLISTIC_MISSILE *bmptr;
BALLISTIC_SUB_MUN *sub_mun;
REAL              speed;
{
    VECTOR impact_pt;
    VECTOR displacement;

      int   i;
      float data_tmp;
      char  descript[80];
      FILE  *fp;

      P(printf("$$$$ M73 file data $$$$\n"););

/*  DEFAULT CHARACTERISTICS DATA FOR sub_m73.c READ FROM FILE          */
      fp = fopen("/simnet/data/sub_m73.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/sub_m73.d\n");
            exit();
      }

      rewind(fp);

      /*    Read array data    */
      i=0;

      while(fscanf(fp,"%f", &data_tmp) != EOF){
            sub_m73_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("sub_m73_char(%3d) is%11.3f %s", i, sub_m73_char[i],
                        descript););
            ++i;
      }

      fclose(fp);
/*  END DEFAULT CHARACTERISTICS DATA FOR sub_m73.c READ FROM FILE */


    bmptr->time = 0;
    sub_mun->impact.timer = 0;
    sub_mun->impact.distance = speed;   /* distance rocket travelled last
                                           frame, i.e. before detonation */

 /*
```

APPENDIX P - sub_m73.c

```
 * get point under sub-munition release point
 */
    impact_pt[X] = bmptr->location[X];
    impact_pt[Y] = bmptr->location[Y] - 10;
    impact_pt[Z] = 10.0;
    missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                            sub_mun, SUB_MUN_IMPACT,
                                            impact_pt, zero_velocity );

}


/*******************************************************************
 * ROUTINE: missile_m73_drop                                       *
 * PARAMETERS:     bmptr - Pointer to a _BALLISTIC_MISSILE_        *
 *                         structure that's ammo-type is MPSM      *
 *                         i.e. it releases sub-munitions of type  *
 *                         _munition_US_M73_.                      *
 *               sub_mun - Pointer to sub-munition structure       *
 *                         associated with _bmptr_.                *
 * RETURNS:        TRUE if time of drop has been long enough to    *
 *                 cause sub-munitions to hit the ground.          *
 *                 FALSE otherwise.                                *
 * PURPOSE:        Simulation of the dropping of munition-type     *
 *                 _munition_US_M73_ rounds.                       *
 *******************************************************************/

static int traj_up = TRUE;    /* TRUE: vector UP -- FALSE: vector down */

int missile_m73_drop( bmptr, sub_mun )
BALLISTIC_MISSILE *bmptr;
BALLISTIC_SUB_MUN *sub_mun;
{
    BALLISTIC_IMPACT *impact;
    VECTOR impact_pt;

    impact = &(sub_mun->impact);
    if( impact->timer == 0 )
    {
        if( missile_util_comm_check_sub_mun( bmptr, MSL_TYPE_BALLISTIC,
                                             sub_mun, SUB_MUN_IMPACT ))
        {
            if( impact->distance > 0.0 )
                impact->timer = (int)
                    ((8 * scaled_rand()) + 1.0 +
                     (sqrt((1.9 * impact->distance) / sub_m73_char[0])));
            else
                impact->timer = -1;
#if DEBUG
            printf( "Height %1.4lf  Time %d\n",
                    impact->distance, impact->timer);
#endif
        }
        else
        {
            impact_pt[X] = bmptr->location[X];
```

APPENDIX P - sub_m73.c

```
            impact_pt[Y] = bmptr->location[Y] - 10;
            if( traj_up )
                impact_pt[Z] = bmptr->location[Z] + impact->distance;
            else
                impact_pt[Z] = 10;
            traj_up = ( ! traj_up );
            missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                                    sub_mun, SUB_MUN_IMPACT,
                                                    impact_pt, zero_velocity );

        }
        return( FALSE );
    }
    else
    {
        if( bmptr->time < impact->timer )       /* wait until sub_mun's */
        {                                       /* hit the ground....   */
            bmptr->time += 1;                   /* incr time counter    */
            return( FALSE );
        }
        else                                    /*   ie.  time == timer */
        {
            if( impact->timer > 0 )
            {
                missile_m73_get_impact( bmptr->location, impact_pt,
                                        bmptr->launcher_C_world,
                                        impact->distance );
                missile_util_comm_release_sub_munition
                    ( bmptr, MSL_TYPE_BALLISTIC, sub_mun,
                     SUB_MUN_IMPACT, impact_pt, zero_velocity );
            }
/* reset time counter */
            bmptr->time = 0;
            return( TRUE );
        }
    }
}


/******************************************************************
 * ROUTINE: missile_m73_impact                                    *
 * PARAMETERS:    bmptr - Pointer to a _BALLISTIC_MISSILE_         *
 *                        structure that's ammo-type is MPSM      *
 *                        i.e. it releases sub-munitions of type  *
 *                        _munition_US_M73_.                      *
 *              sub_mun - Pointer to sub-munition structure       *
 *                        associated with _bmptr_.                *
 * RETURNS:     FALSE if all m73 have impacted the ground.        *
 * PURPOSE:     Simulation of _munition_US_M73_ impacts.          *
 ******************************************************************/

int missile_m73_impact( bmptr, sub_mun )
BALLISTIC_MISSILE *bmptr;
BALLISTIC_SUB_MUN *sub_mun;
{
    BALLISTIC_IMPACT    *impact;
```

APPENDIX P - sub_m73.c

```
    VECTOR              impact_pt;

    impact = &(sub_mun->impact);
    if( impact->timer < 0 )
    {
#if DEBUG
        printf( "ignore under ground detonation\n", bmptr->missile_id );
#endif
        return( FALSE );
    }
    if( bmptr->time < 1 )
        impact->delay = 0;
    else                                /* 0 - 0.250 sec delay */
        impact->delay = (int)(250 * scaled_rand());

    bmptr->time += 1;
    if( missile_util_comm_check_sub_mun( bmptr, MSL_TYPE_BALLISTIC,
                                        sub_mun, SUB_MUN_IMPACT ))
    {
/*
 * send _impact_ to util_ball & to world
 *      missile_util_comm_impact_ball_sub_munition( bmptr, impact );
 */
        impact->quantity -= 1;
/*
 *  get  NEXT  M73  _impact_location_   OR  stop
 */
        if( impact->quantity > 0 )
        {
            missile_m73_get_impact( bmptr->location, impact_pt,
                                    bmptr->launcher_C_world,
                                    impact->distance );
            missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                                    sub_mun, SUB_MUN_IMPACT,
                                                    impact_pt, zero_velocity );
            return( TRUE );
        }
        else
            return( FALSE );
    }
    else        /* Didn't get an impact */
    {
        missile_m73_get_impact( bmptr->location, impact_pt,
                                bmptr->launcher_C_world,
                                impact->distance );
        missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                                sub_mun, SUB_MUN_IMPACT,
                                                impact_pt, zero_velocity );
        if( bmptr->time > impact->timer )    /* time's up */
        {
            printf( "M73_SIMUL timed-out: %d non-impacts\n",
                    impact->quantity );
            return( FALSE );
        }
        return( TRUE );   /* keep trying */
```

```
    }
}


static void missile_m73_get_impact( release_pt, impact_pt, mCw, height )
VECTOR release_pt;
VECTOR impact_pt;
T_MAT_PTR mCw;
REAL height;
{
    VECTOR detonation;          /* Offset Vector in World Coords
                                   of detonation point  */

    REAL   x, y;

    x = height * sin(deg_to_rad( sub_m73_char[1] * (0.50 - scaled_rand())));
    y = height * sin(deg_to_rad( sub_m73_char[2] * (0.50 - scaled_rand())));
    detonation[X] = x * mCw[0][0] - y * mCw[0][1];
    detonation[Y] = y * mCw[0][0] + x * mCw[0][1];
    detonation[Z] = - height;
/*
 * Stretch _detonation_ vector to ensure intersection with ground/vehicle
 */
    vec_scale( detonation, 1.5, detonation );
/*
 * add to _release_pt_ to get location of _impact_ in World Coords
 */
    vec_add( release_pt, detonation, impact_pt );
}
```